UNIVERSITY OF CALIFORNIA,
IRVINE

Energy Optimization for Two-Dimensional NoCs Using Genetic Algorithms

THESIS

submitted in partial satisfaction of the requirements
for the degree of

MASTERS OF SCIENCE

in Computer Engineering

by

Zahraa A M R H Marafie

Thesis Committee:
Professor Nader Bagherzadeh, Chair
Professor Jean-Luc Gaudiot
Professor Pai Chou

2016

DEDICATION

I dedicate my thesis work to my beloved husband Ayoub and my cherished little angels, Alaa and Esraa, for the boundless love, considerate support, ceaselessly encouragements, and the timeless care they showed me during those I consider triumphing days. My words may certainly not be adequate to tell you how thankful I am.

Likewise, I dedicate my thesis work to my loving family and treasured friends. Mom and Dad, thank you for your prayers every night, for the determination and courage you passed to me, and for the faith and beliefs you skilled me with. My dearly brothers, sisters, and in-laws, thank you for being such everlastingly companions within my heart. I joyfully sensed your love, care, and support over the distances. My best friends overseas, Dhoha Marafie, Mariam Alsalem, Fatma Alsultan, Safia Malallah, Mariam Almusallam, Huda Abdullah, Esraa Aljeri, Dana Albahar, Fajer Alfarhan, and Haifaa Alfuziea, I truthfully recognize how prominently you were supporting me. My heavenly wondrous friends, I thank you for offering me strength, confidence, and certainty.

I also dedicate my work to my much-loved UCI friends. Atusa Nasiri, Anastasia Shuba, Maryam SayyedHosseini, and Ali Muzaffar, it meant a lot to be friends with such wonderful beings like you are. Your friendship is the blessing I handheld to endure my research journey, and your presence had always presented relief and support.

Further, I dedicate my thesis work to my undergrad professors whom I name my life mentors, who supported my erudition, cognizance, prudence, insight, and wisdom. You have been watching over me and leading me towards the ultimate purposes of life and existence for many years. No words can contain your indefinite incidence in my life. Thank you for always being there for me, Prof. Mostafa Abd-El-Barr, Prof. Muhammed Sarfraz, Prof. Jehad Al-Dallal, Prof. Helal Al-Hammadi, Prof. Paul Manuel, Prof. Kalim Quraishi, Prof. Hanady Abdul-Salam, Prof. Naelah Al-Dabous, and Prof. Yousef Salem.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ALGORITHMS

ACKNOWLEDGMENTS

I would like to thank my advisor Professor Nader Bagherzadeh for his never-ending support throughout my study and research. Professor Bagherzadeh was always present to direct me and he was there for me whenever I ran into any circumstance that I needed help with. No words are sufficient enough to thank him for all the encouragement, reassurance, and support.

Additionally, I acknowledge my teammate, Mahdi Torabzadeh's role in directing me throughout my work, and supporting me with the knowledge and assistance whenever I looked-for. Mahdi has spent many hours throughout the last year directing me and adjusting the path of this research work. I truly appreciate every single minute of his time and I am very thankful to have such a knowledgeable, intelligent teammate as he is.

ABSTRACT OF THE THESIS

Energy Optimization for Two-Dimensional NoCs Using Genetic Algorithms

By

Zahraa A M R H Marafie

Masters of Science in Computer Engineering

University of California, Irvine, 2016

Professor Nader Bagherzadeh, Chair

The steadfast development of the computers world kept marching along with Moore's predictions in the last two decades. Concurring to Moore's prediction, more transistors result in gaining greater speed. This great speed comes with the trade-off of producing high heat. This prediction has eventually reached to a wall that cannot be crossed unless new technologies are discovered because the heat issues became uncontrollable. One of the greatest discoveries to get over this wall is the NoC infrastructure, which was presented by Benini [4] in 2002. This technology defines a practical solution to improve the energy efficiency and performance.

The inspiration for this work came from Ogras's work in [27], where performance is enhanced for the application-specific NoC-based SoC by adding extra long-range links to two-dimensional mesh topologies. The main focus in this work is to improve the energy efficiency for a general purpose NoC-based SoC by finding the best possible extra links to add to a two-dimensional mesh topology via genetic algorithms. In the genetic algorithm, extra links are added randomly to form the different solutions for this NP-Hard problem. Comparing the energy consumption results of the new NoC design to the regular mesh topology, an improvement of 19% in energy per throughput is obtained. Ultimately, it was found that the more and the longer the links, the higher energy efficiency is achieved.

Chapter 1

Introduction

The realm of computers has changed the worldview into a far-reaching state of an endless inevitable growth of electronic devices that grasped our world and convoyed our lives to become highly essentials. All those devices rest under the term "computer", which was first chronicled in 1613 to define a man who is accountable for mathematical calculations and computations; while this term remained unbroken, the introduction of machines in the 19th century redeployed the name computer to apprehend machines rather than human beings [11]. Around 1822, the conception of the Difference Engine succeeded to chiefly define the first automatic computing machine; an effective machine to compute numerous series of numbers and have the results presented on hard copies developed by Charles Babbage. Later in 1837, Babbage's consistent grind work resulted in the revealing of the Analytical Engine, which design entailed underneath an Arithmetic Logic Unit (ALU), a basic flow control, and an integrated memory, to circumscribe the first general-purpose computer facet. Babbage could not build his machine when he was alive, unfortunately, for of lack of funds, but following in 1910, his youngest son Henry Babbage worked on constructing part of the machine to attain basic calculations [30]. German Konrad Zuse designed and developed his Z1 machine between 1936-1938, which is considered as the very first modern functional

electro-mechanical binary programmable computer. Around the same time, Alan Turing, became a well-known scientist for setting the groundwork and foundation for computing and computer theories we obey nowadays by the invention of the Turing machine. The machine was developed around 1936 and defined by a simple mathematical model to print representations on paper tape and emulate algorithm's logic [28].

The development of modern computer machines continued successfully throughout the years, and every uncovered discovery has accomplished its designated, even though considered simple, purposes. Embedded in every computer machine, there were sets of vacuum tubes invented by Ambrose Fleming in 1904 to convert from alternating current signals into direct current. However, vacuum tubes were measured to be extremely inefficient, as they require reasonably large area, and also because vacuum tubes encounter faults and needed to be replaced rather frequently. Moreover, the computers of the 40s and 50s required being in continuously cooled rooms, to drop the heat produced by around 18,000 tubes installed in each of them [28].

1.1 The Era of Transistors

In November 1947 at Bell Labs, John Bardeen, Walter Brattain and William Shockley moved the world of computers to a new era of remarkably advanced and steadfast developments, which is aimed to their invention of the transistor. The transistor is their great discovery that substituted the vacuum tubes and became enclosed in every electronic device we use nowadays [7].

Shockley had to depart Bell Labs because of some conflicts and went back to his hometown Palo Alto opening his own Shockley Semiconductor Laboratory of Beckman Instruments. Shockley then tried to temp some of his bygone teammates to join him, but his attempts

went unsuccessfully; so, he decided to hunt universities for their insightful and brightest graduates to formulate the beginning of Silicon Valley. While Shockley was marvelous in engaging preeminent people to his company, his coarse management practices failed to grasp his employees. Eight of Shockley's co-workers, later known as the traitorous eight, were distraught because of Shockley's behaviors and chose to resign and later created Fairchild Semiconductor Company, which is the ancestor of almost all semiconductor companies operating nowadays. Fairchild Semiconductor division established its visions to develop silicon-based transistors instead of germanium transistors, where germanium was the common material to build semiconductors. Moreover, the major cost of building the silicon transistors will remain around the manufacturing process, as the material will consist of sand and a few fine wires. The belief of departing the age of disposable appliances and machines was brought to attention, since low-cost electronic constituents used would be inconvenient to repair, and most likely shall be discarded after its been damaged or no longer usable.

Transistor technologies kept developing with innovation to reach to where it is nowadays, and this entire emergence of transistor discoveries lead to having more powerful cellphones than old days supercomputers, and arisen superior personal computers comparatively to the earlier ones. Now, we have cars with dozens of microprocessors, online shopping websites, electronic books on Kindles and iPads, video games that are more powerful than flight simulators of two decades back. The invention and advances of transistors and the integrated circuits are two major advancements, which appraised the successful growth of electronics.

1.2 The Discovery of Integrated Circuits

The problems encountered by the vacuum tubes were no longer present with the changeover to transistors. Transistors their own had issues that arise from a different scope. The need to solder the transistors together resulted in a farther complex circuit with manifold high-

complex connections linking the transistor components together, thus, inflating the tendency to have defective wiring.

Humanity is fortuitous for having Jack St. Clair Kilby of Texas Instruments to develop and manufacture the first integrated circuit, so-called a chip, which represents a collection of minuscule transistors that are linked during the manufacturing phase; hence, the requirement to solder transistors together was sensibly invalidated; leading to mainly consider the required connections between the electronic components of the circuit rather than worrying about other obstacles. The Integrated Circuits reduced area used to connect transistors and other components that lead to shrink the distance that the electrons had to travel, which eventually moved forward to reach higher-speed machines.

1.3 Moore's Law

Four decades ago, on 16 April 1965, Gorden E. Moore published on Electronics Magazine his prediction that the number of transistors in the electronic devices is steadily doubling as shown in the Figure 1.1, approximately, every couple of years [25, 24]. Conformed to Moore's Law, there has been vast development in the semiconductors industry leading to extensive growth in the transistor counts of the semiconductor devices. Moore stated that he could never see more than the next couple of chip generations, and after that it looked like we'd hit some kind of a wall [1]. Such pessimistic prediction was logical to Moore because of the exponential increase in the transistor counts.

The breakdown of Moore's Law seems more imminent than before, as it's getting more challenging to balance the tremendous expansion of the numbers of transistors in the electronic devices and the energy to be consumed. The conventional machinery has a power dissipation of around $2.5 - 5fJ$ per bit flip. An Intel Pentium IV processor with 10percent activity has

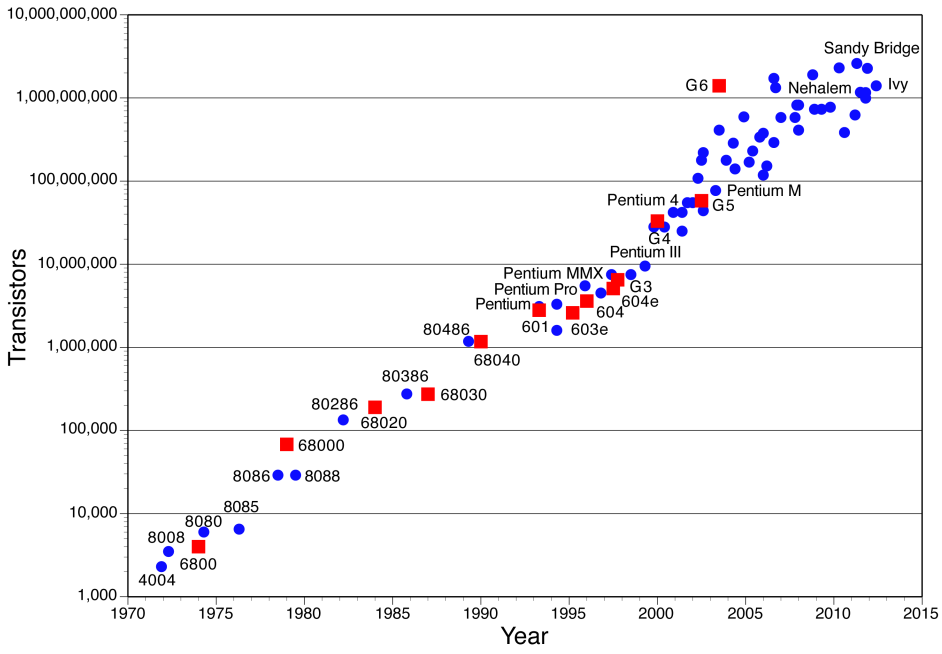


Figure 1.1: Moore's Law: Number of transistors doubles every 18 months

a power dissipation of $50 - 100W$ per cm2. Assuming a 10GHz operation in a transistor with 1013 count and a $1fJ$ bit flip energy, then the power dissipation shoots up to $100MW$ per cm^2, which is equivalent to the heat of a rocket nozzle [8]. The International Technology Roadmap for Semiconductors (ITRS) terms this as the Red Brick Wall that cannot be crossed [2]. This spells doom for Moore's Law. Hence we face an inevitable meltdown if we continue to tread down the path we are proceeding to follow nowadays.

1.4 Introduction to System-on-Chip (SoC)

The steadfast and immense developments in the silicon technologies reached to a point where we can sense billion-transistor chips; such single chip ICs archetypally include multiple complex heterogeneous constituents, such as, the programmable processors, on-chip memories, I/O interfaces, and communication architectures that functions mainly as an interconnection structure to serve different components' communications [29].

SoC is a design methodology of the challenging problems to integrate the electronic components of a system into a single silicon chip. In the old days integrated circuits used to carry responsibilities of performing only simple operations, like, decoding and encoding operations, A/D and D/A conversions, and similar simple processes. Technology developments and expansions never ceased, and additional functionalities were incorporated and integrated into a single chip with time. Consequently, a chip arisen to be capable of performing the operations and procedures of a complete electronic system, such as, a network router or an MPEG decoder, that ultimately resulted in the definition of the SoC for those chips holding an entire system means. The SoC designs have shown less power consumption, better reliability and estimated less costs, due to their shorter connecting wires and their high level of integration. Nevertheless, SoCs assembly costs are low because the assembly of SoC requires fewer packages to combine and connect. Yet, the success of SoCs depends greatly on selecting the suitable design, using the right process technologies, and also, its ability to interconnect exiting modules in a plug-and-play fashion [36, 31].

1.4.1 SoC Technology Challenges

The new revolution of SoC solved a lot of complications confronting the silicon technology; conversely, those solutions never came free as SoCs brought up new challenges to work on. SoC synchronization using a sole clock source is most likely impossible to implement, accordingly, globally asynchronous and locally synchronous synchronization paradigm is the forthcoming vision for SoCs. A globally asynchronous and locally synchronous system is generally considered a distributed system with no sense of global coordination, and that would cause difficulty in controlling the information traffic globally.

Authors of [4] state that the progress of SoC evolution keeps its complexity scaling; accordingly, affecting the ability of the system to operate in a completely deterministic manner.

Communication synchronization downfalls are expected in spite of their unusual occurrences. Additionally, there are possible minor logic strikes, which are most possibly lower than one volt, and can be triggered by energy usage and device reliability. Besides, data errors are likely to be generated because of the electrical noise, radiation-provoked charge injection, and electromagnetic interfering. Therefore, the normal way of transmitting data via wires will very undoubtedly be unreliable and most importantly non-deterministic. Generally, deterministic and stochastic models are combined to form the ground of the SoC design methodologies forming a modular component-based approach that satisfy together software and hardware design. Benini sees that the layered design used in re-configurable micro-networks is the most efficient and resourceful communication network for future SoCs.

1.4.2 Network-on-Chip (NoC) Design Approach

In 2002, Benini [4] proposed a new approach derived from models for large-scale designs, where in this methodology SoC is considered a set of micro-network components, and its network defines the communication medium for the components, with respect to quality-of-service specifications and requirements. By using the micro-network stack paradigm showed in 1.2, Benini described the electrical logic abstraction, all functional properties, and the interconnection fabrication.

SoCs provide smaller gates and memory cells that are considered an advantage for SoC design in terms of communication energy reduction. Reducing the global communication energy use is an emergent concern, as the projections of the delay optimization methods of global wiring predicted that on-chip communication would definitely entail greater amounts of energy. Furthermore, monitoring and controlling the network traffic may assist in improving the power management for the communicating computational resources.

Moreover, SoC network design-time specialization rise a lot of challenging problems, as com-

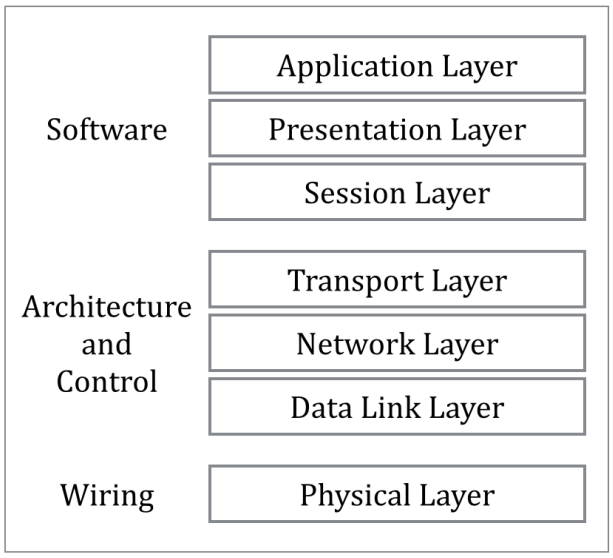


Figure 1.2: Micro-network Stack Layers

munication network design has been conventionally disjointed from applications and became extremely subjective to the standardization constraints of communicating networks, while in SoC networks, designers propose the communication network framework from scratch. Accordingly, standardization has to be applied solitary to the abstract network-interface for the end-nodes, and network architecture can be personalized according the design needs. Benini had foreseen a vertical design-flow derived from the micro-network layers, where each layer is specialized for the aimed application region, and from that perspective, a widespread inspiring research field has uncovered its gates [4].

1.5 Interconnection Networks

Multi-processor parallel computers interconnection networks and on-chip networks are comparably related considering each separate chip as a singular processor, and similarly nodes are located close to one another illustrating high reliability connection links. Additionally, interconnections between multiprocessors have been developed with extremely stern latency and bandwidth constraints to maintain successful parallelization, and equal restrictions can

be applied to the micro-network design of the SoC [20, 4].

1.5.1 Shared-Medium Networks

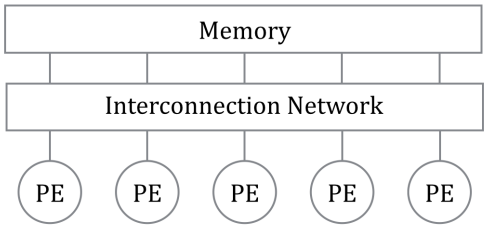


Figure 1.3: Shared-Medium Interconnection Network

The majority of SoCs present nowadays use the very basic interconnection structure, the shared-medium architecture, which embraces a transmission medium to allow communication between its devices. In such networks, only a single device is permitted to take control over the network freely. Also, when information stream from less numbers of transmitters to more numbers of receivers causing greatly asymmetric communication to happen, shared medium networks are capable of supporting such broadcast. Moreover, the backbone bus is believed to be the most popular on-chip shared-memory structures that provide convenience and low over-heated interconnections. Such structures can manage a limited number of active bus masters along with various passive bus slaves who mainly answer master requests. Thus, bus arbitration approaches are required when multiple processors attempt to acquire the bus at the same time.

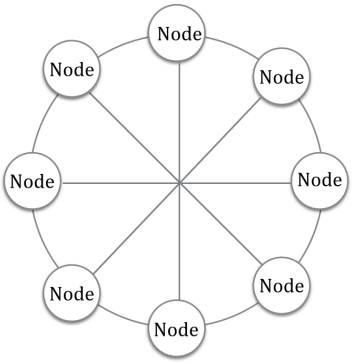
Using a bus arbiter module that carry out centralized arbitration, and any processor shall demand a bus mastership from the arbiter in order to issue communication. Such procedure necessitates high speed and low occurrences since loss of communication performance can arise. Along with the arbitration, severe performance loss may occur because the master has to wait for the slaves to reply back. By using split-transaction protocols, the bandwidth can be minimized and the chances of such complications remain absent. In this protocol, a bus

mastership is released once the request is complete, and salves shall gain access to the bus in order to show its response [4].

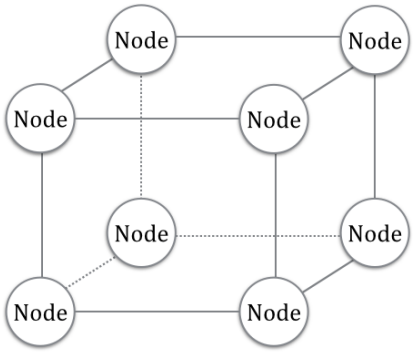
Shared-medium architectures are well known and are used broadly, but they lack in scalability measures. The conventional bus used in SoCs remains convenient to integrate no more than five processors, but it cannot handle anything more than that. Furthermore, shared-bus is generally energy inefficient, as any data transmission has to be broadcast in order to reach each likely receiver with great energy overheads. Ultimately, future systems will hold even hundreds of processing elements that generates lots of data and information to be transferred, and bus-based networks will not be able to manage the bottlenecks of power and performance [4].

1.5.2 Direct and Indirect Networks

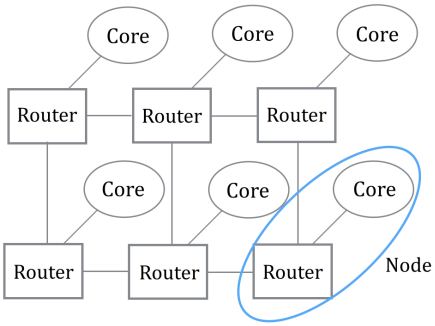
A solution to network scalability issues can be found in implementing the point-to-point networks, where nodes are connected straight to a certain number of immediate neighbor nodes. Those nodes are connected to each other through a router, which is mainly responsible for the communication between those directly adjacent processing elements. The direct interconnect networks are used widely for constructing large-scale systems, and the more nodes the system contain, the higher communication bandwidth is. On the other hand, the indirect interconnect networks propose a different network solution where connections between nodes is directed via switches. Each node is coupled with a network adapter that is responsible for offering a programmable connection between its different ports, while the switches define the programmable connection between those ports. Conclusively, the difference between direct and indirect networks is becoming unclear as the structures become more complex and functionalities of each is becoming part of the other's.



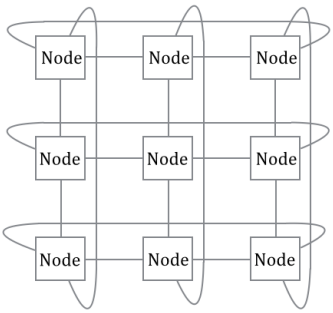
(a) Octagon



(b) Hypercube



(c) 2D Grid

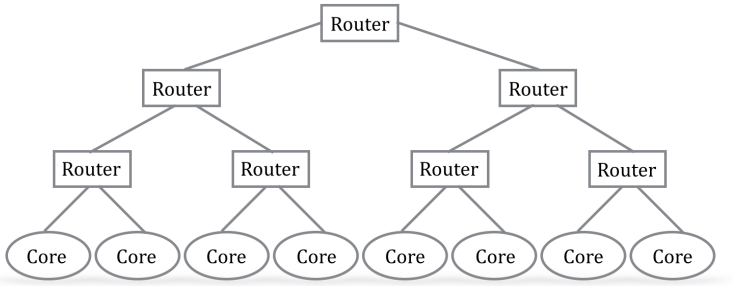


(d) Torus

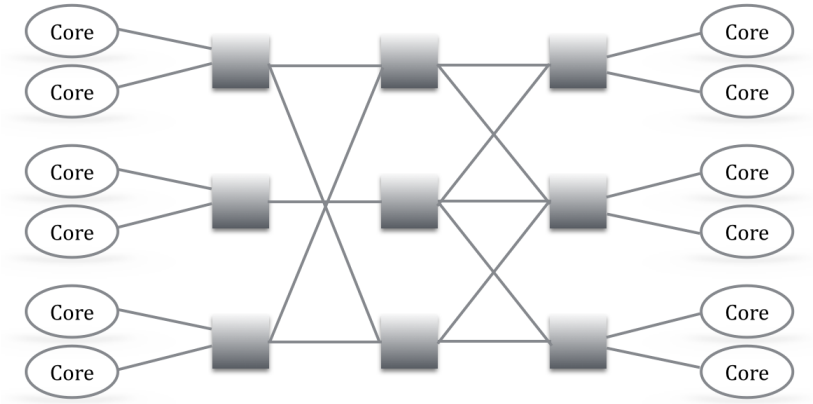
Figure 1.4: Direct Topologies

1.5.3 Hybrid Networks

Normally in hybrid architectures, high communication bandwidth processing units and lower bandwidth inter-cluster communication links. Moreover, hybrid networks use a very small part of energy and communication resources to provide better performance in comparison to homogeneous networks, and because of that, it is highly preferable to be used in the different systems [4].



(a) Fat Tree



(b) Butterfly

Figure 1.5: Indirect Topologies

1.6 Micro-Network Layers

The micro-networks depend mainly on network control algorithms, where network control manages the network resources dynamically during system run-time in order to provide the necessary QoS. Further, NoC is considered a packet-switched network, and the idea of it is derived from the parallel computing. By applying networking concepts to on-chip networks, the communication is decomposed into the seven layers of the ISO/OSI model as shown in Figure 1.2. Normally, the model performs as an abstract to follow, rather than a complete definition for the architecture of a system. The model allows communication between neighboring layers while each layer cover its complexity from atop layers [5]. In order to explain the network communication concepts for a system using the micro-network

stack, it is important to understand the three layers or Architecture and Control. (Refer to Figure 1.2)

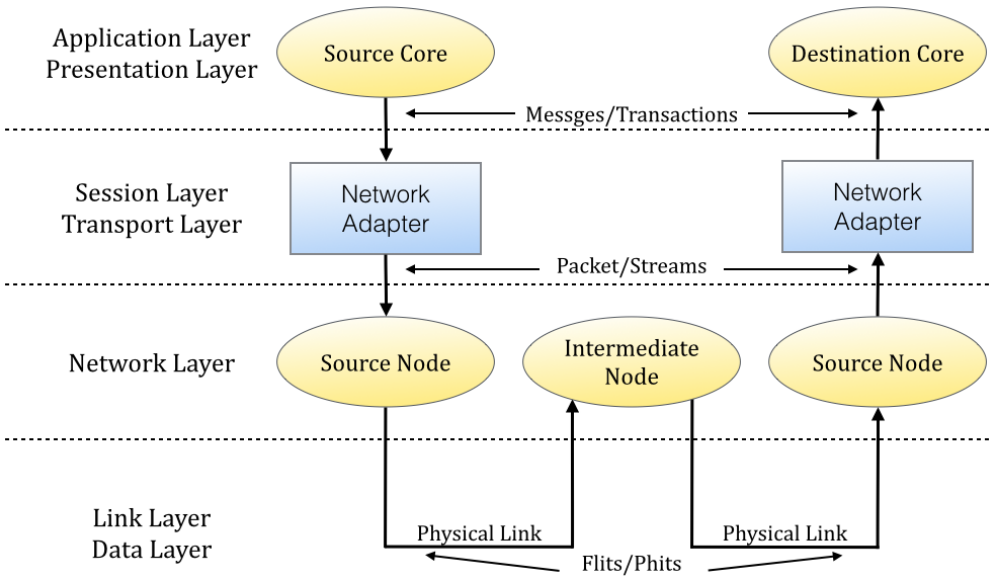


Figure 1.6: ISO Reference Model for NoCs [33]

1.6.1 Data link layer

Typically, disputation in a shared-medium network brings up an extra overhead of errors. Also, resolution of the disputation is a non-deterministic procedure, which brings forth further noise to the network, as it requires synchronization between the nodes in the system. Generally, non-determinism can be eliminated through Synchronization that comes at a price of performance.

Using packets to send data through a network is extremely useful to reduce communication faults. Besides, when packets are sent over a channel with low reliability, then surrounding the errors within the packets will be a lot easier. Packets have boundaries, which help contain the errors and fix them in a packet-by-packet manner. Additionally, error-correction codes can be used to succeed in correcting errors by adding redundancy to the transferred data with a balance from packet-based error-detection and -recovery protocols. There are

numerous parameters in such protocols that can be modified in order to reach the highest possible performance at a certain error probability and particular energy intake [4].

1.6.2 Network layer

Typically, the network layer is responsible for end-to-end delivery control in networks containing many communication channels. Current NoC allow the processing elements to be connected to the on-chip bus causing to have an blank network-layer, yet, when we have a set of links to connect the processing elements together, a decision must be taken in order to set-up the connections between consecutive links and direct the data from the source to destination. Studies on switching and routing have gone broadly and reached great accomplishments in the frameworks of communication for both general networks and multiprocessors.

Switching in Networks

There are mainly three classes of switching, and those are: packet switching, circuit switching, and cut-through switching [20]. These switching techniques provide better channel utilization and delivery time when predictability rates drop and variance rates rise. Research showed that cut-through switching is most likely the preferred on-chip micro-network in terms of performance measures. High forwarding data rates across switches may escalate the risks of contention and increase traffic rates that lead to energy being misspent.

Routing in Networks

Routing is usually tightly coupled with switching, where routing algorithms construct a route for the packets to move through within the network until it reaches its end destination. On-

chip routing algorithms are evaluated and classified according to the following trade-offs:

- Average performance traded off with predictability
- Speed and complexity of routers versus possible channel utilization
- Robustness of an algorithm against aggressiveness

Determinism versus Non-determinism in Routing

There is a great difference between deterministic and non-deterministic routing algorithms. Deterministic techniques normally stream the same path between a assumed pairs of source/destination to provide the finest choice for uniform traffic/ regular traffics. On the other hand, the non-deterministic adaptive routing works through using current information of network traffic in order to avoid congestion areas. Also, the adaptive approaches are highly preferred, especially when operating with irregular traffic patterns or nodes/links with low reliability issues [4].

Authors of [4] estimate that the future on-chip micro-networks designs has to highlight the speed and the decentralization of the routing choices, along with fault-tolerance and robustness. Such aspects, besides the irregularity of special-purpose SoCs support then need for adaptive routing rather than deterministic routing. Yet, in some systems where traffic patterns are highly predictable, then deterministic routing may be a satisfactory choice.

1.6.3 Transport layer

The transport layer lies above on the network layer, and it works on packetizing the messages into small packets. Also, once the message reaches the destination, it is resembled and sequenced by the transport layer. The granularity of packets is a serious design issue,

because packet size needs to be considered in most control algorithms. Packets are normally standardized in most networks, but packet standardization can be changed and customized in SoC micro-networks at the design phase.

Generally, flow control basis is provided by either statistical or deterministic techniques. Statistical procedures provide better resource utilization than what deterministic gives; yet, they cannot satisfy worst-case assurances. On the other hand, deterministic methods certify that traffic specifications are met, and are able to provide solid boundaries for chances of message lose or delay. Conversely, such techniques carry the disadvantage for being established on worst cases, but ultimately, they lead to substantial under-utilization of the network resources.

1.7 Traffic Modeling

One of the very important challenges in the computer architecture design is the measurement and comparison of cost, performance, and other architecture parameters. The formal description of NoC architecture relay mainly on the transitions and states in comparison to the states that are caused by the distinctive actions of the system. Those models signify an abstraction of the system's behavior that is adequately accurate for examination and evaluation, and is convenient for verification purposes. However, the available conventional benchmarks are application-specific, and cannot be used for measuring the highly intense communications in a NoC. Besides, the existing SoC benchmark circuits contain limited number of blocks, which does not work on the scalable NoC-based architectures. Therefore, as traffic in the NoC varies broadly throughout execution process, new advanced traffic models were required to feature the aspects of behavior of the NoC systems [33].

The traffic models can be categorized to either synthetic or realistic traffic models. Basically,

the realistic traffic models are normally traces of application executions on the NoCs, while, the synthetic traffic models are arrangements that communicate the abstract models of the packets delivered and received to/from different nodes. Realistic traffic illustrates an explicit description of applications, while synthetic traffic is normally generated according to applied mathematical models. Henceforth, synthetic models conceal a wider range of applications executed onto the NoC platforms. It is a great feature of the synthetic traffic model as it allows a large continues stress on the network with regular or predictable patterns. Because those traffic patterns do not really represent a true application, it cannot be retained for correct and truthful design exploration space, whenever an application specific NoC platform is required for design [33]. Synthetic Traffic models can be classified to either Temporal or Spacial models described below:

1.7.1 Synthetic Traffic: Temporal Distribution

Temporal distribution typically defines the way each individual node work to produce traffic throughout time and this traffic is spread on the NoC architecture. There is a list of traffic properties that are incorporated with the temporal distribution, such as, the rate of generated messages, the timing information. Each property can be applied using values that can be periodic, random, normal, or any other possible form [33]. There are many temporal distribution models, and below is a description of a few well-defined models:

Perfect Synchronous

The perfect synchronous model is a time-triggered model, which assumes that no computation or communication take time to finish, and that is based on the Perfect Synchrony Hypothesis. To be more specific, to have synchronicity is employed perfectly; it means that once an input is given, the output of the computation is given at the same instance. More-

over, when several processes are interconnected, then the results of computations will flow instantaneously throughout the system. Such model is surely easy to understand and implement. The analysis cannot come out perfect when operating the system with this model, since no timing information is involved. Inferences resultant by the applying this approach can have a huge difference in comparison to other approaches, which are considered more accurate [33].

Clocked Synchronous

The Synchronous Hypothesis Assumption grounds this approach, which is also a time-triggered model, where it is assumed that each computation in the system is controlled by the global clock signal. Usually, in the clocked synchronous model, the computation requires one clock cycle to complete, while the communication needs no time; as a result, this approach is considered as a great option for the cycle-true models. Insufficient consideration for real physical time is an obvious drawback of such system [33].

Discrete Events Model

This model is a contrast of time-driven models, as it considers the events taking place in a system, and actions are taken depending on the events of the discrete event model. Discrete events model allows using floating-point time expressions, which in turns, allows simulating physical time and simplify the simulation procedure. Conversely, performance of the system might reduce because of the need to generate events through an event queue, and that is more likely to affect architectures with high switching activity [33].

Models Concerning Packet Generation

According to [33], there are many traffic models that have been designed to handle various cases and situations. Those models mainly focus on packet generation to satisfy the systems communication abstract of packet delivery and reception. One of the simplest and easiest models to discuss is the uniform traffic model, where in this model, it assumed for a network with N nodes to have the probability for a node to send packets to another node in equivalence to the equation $1(N-1)$; with a constraint of that a node does not send any data to itself at any time. A uniform traffic model specifies that the length of the packets, latency between different packets, and the destination of packets are selected in pure randomness according to set utilization ratios that are defined be the designer or even throughout a random procedure.

Another model is the uniform random model, where each node is eligible to produce packets in a random manner according to the probability of $\lambda = 1/N$, where N is the total number of nodes in the system. There are also other models that concentrate on traffic generation in a timely manner with concentration on getting through the shortest possible path, such as the locality traffic model. In such model, a node has a probability of P to send a packet to the destination node and that is reliant on the distance between source and destination.

Moreover, an alternative model that focuses on sending a constant number of its traffic to the neighboring nodes, while the rest of traffic is spread using random and uniform models is the nearest neighbor traffic model. This technique is widely used to evaluate the impact of communication locality on the power consumption and performance of the NoC.

Further, in the HotSpot traffic distribution model, we have N total number of nodes, and $M \in \{2, 4, 8, ..., N\}$, where those parameters are defined by the user. Basically, HotSpot traffic is defined by the selection of $[N \backslash M]^2$. Then, by using an explicitly defined fraction $p \in \{0.5, 0.7\}$ of traffic, chosen hotspots are targeted one at a time via uniform random

selection, while, the rest of the nodes receive traffic uniformly.

The Burst-Mode traffic model is seemly appropriate for the emulation of standard burst modes generated from the physical cores, and it implies that packets can be sent in accordance to a fixed packet generation rate. It is assumed also that no traffic can be moving between nodes for a beforehand-defined period of time when system is at a stable state. Any periods of time of stable or active states are set by the designer and can possibly be changed in convenience of meeting the goals of study of the distinctive utilization issues of the network architecture [33].

1.7.2 Synthetic Traffic: Spacial Distribution

The other category of the synthetic traffic is the spatial distribution traffic models, which also contain many properly designed traffic models. The simplest approach in this category is the uniform traffic model, where each node is permitted to send packets to randomly selected destinations. In contempt of having this model as the simplest, its plainness and simplicity might result in intolerable evaluation outcomes for the NoC architecture.

1.8 Routing Algorithms

Generally, routing associates selecting a path from a source location to destination location in a specific topology. According to [12], it is important to know that routing is considered as one of the significant factors that govern the performance of a network. There are many reasons behind selecting decent routing algorithms, as good routing algorithms balance the load throughout the network channels no matter what traffic pattern is used. Basically, higher load-balanced channels cause the network to closely reach the ideal throughput. It is unexpectedly surprising to know that many of the existing routers handle the load balancing

inadequately and does a poor load balancing job, while satisfying a secondary goal of routing, which is, using shortest paths to reach the designated destination. Designing a routing algorithm with satisfactory characteristics, also support finding a shortest path as much as possible, helps in reducing number or hopes, and works on decreasing the overall latency. Load-balancing and minimal routing are unnoticeably conflicting with each other in terms of supporting maximum throughput.

In oblivious routing algorithms, algorithms work on increasing the average path length in order to enhance the load balance, which make this tradeoff true and existing for oblivious routing. On the other hand, a smart designer will offer an approach that brings up the best of both worlds, so instead of selecting an algorithm that does not consider the traffic pattern as mentioned about oblivious algorithms, why not create an algorithm that is adaptive to the current traffic conditions? Having such algorithm will assist in sending traffic in a minimal approach to support the use of easy traffic patterns as the uniform traffic, while assisting in giving the ability to switch to non-minimal routing for "hard" non-uniform traffic patterns; and that is the basic idea of adaptive routing. If we look at the broad image of those routing algorithms, we can realize that the potential advantage of them is bearing in mind both the load-balancing issues and the minimal paths matters. The design implementation result in issues that make it extremely challenging to satisfy this goal [12].

Furthermore, one of the very important aspects of the routing algorithms is their capability to function during fault occurrence within the network. Assuming that a specific algorithm is hardwired into routers, and a fault occurred in one of the nodes or links causing it to fail; that means the whole system will fail too. Conversely, if an algorithm is reprogrammed to become adaptive to situations of failure, then a system can remain active and running with a little loss in overall performance due to the failure that occurred. It is observable that having such feature for the routing algorithms in critical for the systems that require exceptionally high reliability [12].

1.9 Virtual Channels (VC)

The NoC definition is based on a set of concepts inherited from distributed systems and computer networking theories to satisfy new well-structured and highly scalable methodologies to interconnect the IP cores of a system. NoCs are favorably capable of supporting the needs of the future computer architectures. There are several problems that need to be tackled in order to get the maximum benefit out of this new design. One of the issues that might be faced by the NoC architectures is the Network Congestion, which extremely affects the overall performance of a system. This issue is very noticeable in the networks that have a single buffer affiliated with each input channel. A single buffer can greatly support the router design in terms of simplicity, yet, it blocks the packets from sharing physical channels at any particular instant of time. In [22], the goal of reducing performance penalty resulting from packet concurrence for the network resources in NoCs. By using VCs, a remarkable technology that reduces latency and increases throughput by multiplexing the physical channels, we can reduce congestion in the network. Besides, the enclosure of virtual channels gives the prospect to set policies and rules for allocating the physical channel bandwidth, which conclusively, sustain the quality of service in the applications.

1.10 Genetic Algorithm

The developments of human scientific skills over the history happened through the construction of knowledge that qualifies us to predict motion of planets, climate changes, solar system, disease elaboration, economical issues, language developments, paranormal subjects, and everything that surrounds us. Recently, we came to recognize some fundamental bounds to our abilities to foresee things and predict. Humans have developed progressively complex resources to manage our interactions with life and nature, and we have absorbed what

aspects are controllable and what are not. The rise of electronic computer is said to be a revolutionary development in the history of science.

This ongoing revolution is intensely increasing the humans' capability to predict and control the nature in different means. Many believe that the great achievements of this revolution will be the conception of new species in the form of computer intelligence. The creation of artificial intelligence goes back to the old days, where the earlier computer scientists had great interest in biology and psychology as much as computers and electronics. Nature was the motive and guidelines to achieve their dreams and visions. Computers in the early ages, we not only used for calculations, but also there were many experiments to model the human brain and biological evolutions. The computing activities of that field developed to fork into three different fields. Those fields are: neural networks, machine learning, and evolutionary computation of which genetic algorithms descend from [23].

Chapter 2

Related Work

2.1 It's a Small World after All

According to Umit Ogras [27], the gird-like regular NoC architectures support global interconnections while also keeping well balanced electrical parameters and lower power consumption. On the other hand, such architectures require crisscrossing many hops between two remotely communicated nodes, which alternatively, may increase the chances of message blocking occurrence and leading to unpredictable latencies and inflexibility of service operation. Besides, real-time applications are broadly distinctive in terms of communication needs; for that, application-specific designs are more preferable for their ability to provide clear level of performance. Improvements resulted from having fully customized topologies come with the sacrifice of modifying the regular consistency of the grid structure making it contain variant sizes of global wires. Therefore, complications of cross-talk, time closure disputes, and wire routing specified algorithms could lead to depress the gains anticipated from this customization. Ogras stated that having pure-regular NoC, or entirely customized NoC are not the only conceivable choices to solve the problem, as many other fields' of study use

networking solutions that are neither fully customized nor completely regular [21, 34, 35]. Those networks can be seen as superposition of clusters connected via short-range-links and a few long-range links, which can result in benefit of plummeting the distances between different constituencies in the network. Networks including shorter paths between distantly located nodes confine the primary notion of the small-world phenomenon that is traditionally identified as six degrees of separation, and it is convenient for its ability to yield logarithmic relation between mean inter-node distance and the network dimensional size. [26, 34]

2.2 Modified Mesh NoC

As Ogras pointed out in his paper, he sees a prodigious potential in using the small-world phenomenon within regular mesh NC topology, which can possibly enhance the NoC performance by adding a few additional long-range links to the simple regular NoCs. Those links obviously benefit in decreasing the average distance between remotely located nodes and also improve the network reliability if the links connecting nodes was selected cautiously. Adding extra links to the topology has a pronounced effect on the traffic congestion in the network. Besides, at relatively low traffic loads, the average packet latency displays a feeble dependency on the traffic injection rate, conversely, the packets-transfer latency escalates sharply and network throughput begin to fall once the traffic injection rate surpass a specific critical value. Figure 2.1 shows the left-hand side area, which signifies the low-traffic congestion state "free-state" and the area afar the critical value is the "congested state", where the changeover from one state to another is identified by the phase-transition. By exhibiting the concepts of long-range links to the mesh topology of NoCs, evident delay in the approaching the congestion state is attained, not to mention that a slight shift frontwards in the critical value brings forth valuable decline in the average packet latency and rises the network throughput [16]. Conclusively, Optimizing the network performance in terms of

average packet latency reduction and throughput enhancement is the key objective of adding extra long-range links to the network.

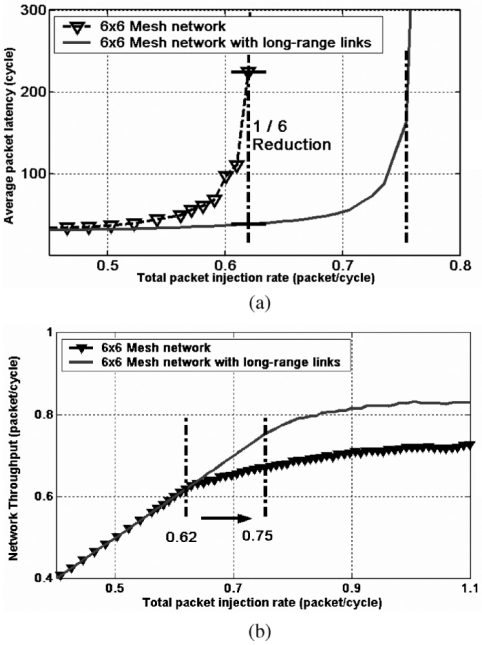


Figure 2.1: Improvement in phase transition region after adding long-range links to the mesh NoC

Ogras stated in his paper that the idea of optimizing application-specific topologies through long-range links was first addressed in his paper, while adding extra links was addressed previously in works related to network theories. As an example, [15] proposed the paper showing the effect of adding random links to a 2D-mesh and torus architectures when its under uniform traffic, while Ogras used in his work application-specific traffic patterns, and also presented an algorithm to insert long-range-links with consideration of traffic patterns. Moreover, since adding additional links may cause a deadlock within the 2D mesh, Ogras presented a deadlock-free algorithm for a 2D mesh with extra links.

2.3 Basic Model and System Assumptions

The model Ogras defined is a set T of mxn tiles interconnected via a 2D mesh network, where the processing elements are able to communicate with each other through the network. The frequency of communication between the nodes is considered in this system, while no assumptions regarding the packet injection rates are made. Furthermore, a worm-whole switching is assumed due to limited on-chip buffer resources, however, the work presented in Ogras paper is applicable to other switching techniques such as packet cut-through switching and virtual cut-through switching. The routing algorithm has to be deadlock free and minimal too in order to satisfy the on-chip requirements. Minimal deadlock-free algorithms are essential for NoCs because recovery mechanisms and deadlock detection methods are overmuch costly and can get the system into to unnecessary delays. Therefor, XY routing is used throughout this work, where the usual routers stay to use the default XY routing algorithm and the routers holding extra links use the proposed deadlock-free algorithm. Additionally, the number of long-range links is limited to only one link per router in order not to exceedingly alter the regularity of the mesh topology. Such constraint brings forth significant performance improvement while employing fewer changes to the actual topology [15].

2.3.1 Long-Range Links Insertion Algorithm

According to Figure 2.2 [15], communication frequencies between the tiles are taken as an input, the default routing algorithm, and also the number of resources allowed to add to the design. The algorithm works on finding the most beneficial link to be inserted to the architecture, and once the link is found, the details of that link is stored and the procedure repeats until all available resources are used. Afterwards, the routing data and the architecture design file will be generated to provide the new configuration.

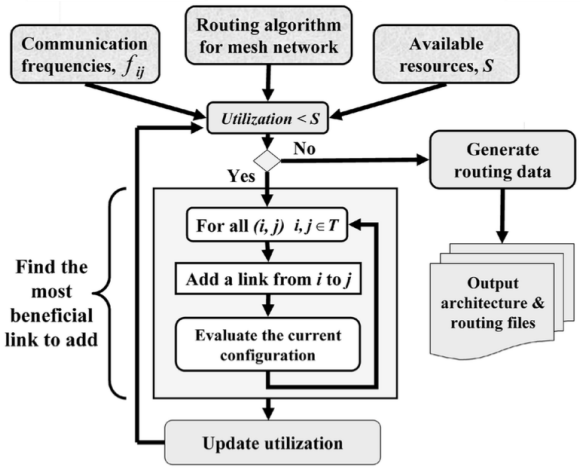


Figure 2.2: Long-Range Links Insertion Algorithm [15]

2.4 Routing with Long-Range Links

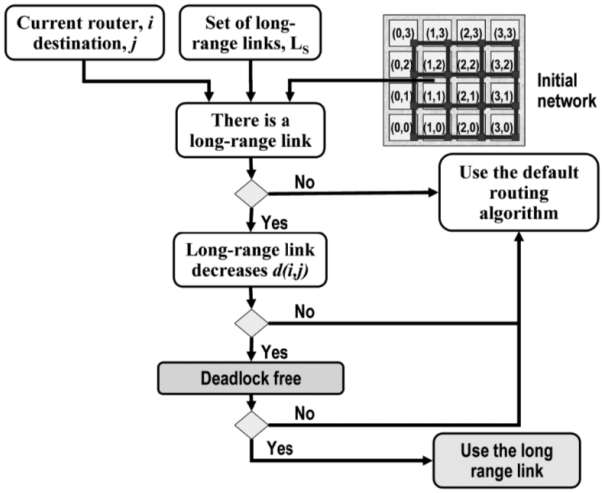


Figure 2.3: Routing Strategy [15]

The paper proposed a routing algorithm that provides minimal paths from source to destination by effectively utilizing the long-range links (See Figure 2.3). First, the algorithms checks if there exist a long-range link on the current node, if not, then use the default algorithms, otherwise, the distance towards the destination node is computed as the following

generalization definition:

$$d(i,j) = \begin{cases} d_M(i,j) & \text{if no long-range link} \\ min(d_M(i,j), 1 + d_M(k,j) & \text{if } l(i,k) \text{ exists.} \end{cases} \tag{2.1}$$

The proposed approach provides scalability and overall improvement for the network dynamics since only local information is used when computing the distances. If a long-range link is present and provides a shorter path to the destination, then the algorithm checks if going through this link may cause a deadlock or not [15]. The paper describes a proof for how it provided a deadlock-free algorithm through the following Theorem:

Theorem 2.1. *The combination of XY routing for the routers without any long-range link, and South-East routing algorithm for the routers with (at most) one long-range link is deadlock- free.*

2.5 Implementation of Long-Range Links

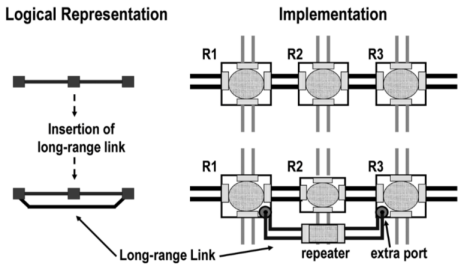


Figure 2.4: Repeaters [15]

Long-range links are divided into segments of regular fixed-length links and attached to each other through repeaters. The repeaters act as simplified routers entailing only two ports, and its job is to receive flits from the in-port, store them in a FIFO manner, and lastly forward

them to the out-port as shown in Figure 2.4. Generally, the repeaters pipeline the long-range links yielding paths that bypass the routers, and look the same as the original routes afforded by the routers. In order to guarantee latency-insensitive operations, then repeaters with at least two-flit buffer size shall be used as described in [9, 10]. It is also important to consider the overhead of adding an additional port to the router for the long-range links. This has been considered by accepting adding only one long-range link per router, which allows minimal alteration to the network regularity while gaining significant improvement.

2.6 Energy Considerations

The proposed approach can be looked at in terms of energy considerations too along with the communication issues. The energy required for transmuting one bit of information through the routers, links, repeaters was measured by:

$$E_{bit} = E_{Lbit} + E_{Bbit} + E_{Sbit} \tag{2.2}$$

where E_{Lbit} defines the energy used by links, E_{Lbit} defines the energy consumed by buffers and E_{Lbit} is the energy used by switches. The energy used by the links never change, because the number of links traversed does not change. The same goes for the buffer when assuming that the identical buffers are used for both routers and repeaters.

In contrast, the switch energy consumption is changed, as some packets will move through the routers sometimes, and may move through the repeaters at other times bypassing several routers. A reduction in communication energy is accomplished by moving through the repeaters with a penalty for the enlarged route size.

The energy consumption had been evaluated using a simulator and a FPGA prototype prior and after adding the long-range links. Results showed that energy consumption of buffers and links have went higher in about 2%, while there was about 7% decline in switches energy consumption. Also, by evaluating the design on Orion model combined with a simulator, it was found that there is about 5% of saving in total energy use. Also, there was slight effect on energy consumption for real FPGA prototypes when adding extra long-range link to the design.

2.7 ReNoC: A NoC Architecture with Re-configurable Topology

The author of [32] discusses re-configuring a NoC topology, where the architecture design is focused on providing a general SoC platform. This general platform can be customized for the application that is currently running on it. The customization may include direct and long links between the IPs. A configuration layer is inserted between the links and routers, and it allows the architecture to use them combination of existing routers on NoC allowing it to become a general purpose architecture, with great considerations of energy-efficiency.

2.7.1 Motivation

Since the majority of research work done on NoCs focus on packet switching, because of its flexibility of allowing the same physical link to be shared by multiple connections. Normally, a general-purpose 2D mesh topology is usually used for such purposes, permitting a packet to pass through several routers for neighboring and distant IPs. Since the foreseen future SoCs may contain hundreds of IP blocks, then a NoC has to support the large number of connections between them. A consequence of having large number of connections is that the

routers have to be fast enough to afford the required bandwidth. Since different applications carry communication constraints, routers become even more convoluted in assisting various Quality-of-Service concerns; which eventually, cause a router to consume the greatest amount of energy among other components. Exploiting knowledge of the application running on the SoC is a major factor in reducing latency and power consumption. Generally, in [27], a few extra long-range links are added to a NoC to decrease latency and improve energy efficiency. The research considers static topologies, and more specifically limits the improvement to only application-specific NoCs. Optimally, using physical circuit switching, countenances direct and more efficient physical connections between the different IP blocks. Mainly, the objective of the proposed ReNoC architecture is to combine the best of worlds of packet switching and physical circuit switching. It is good to mention that this paper was the first paper that has presented such idea of combining those two methods together [32].

2.7.2 ReNoC Basic Components

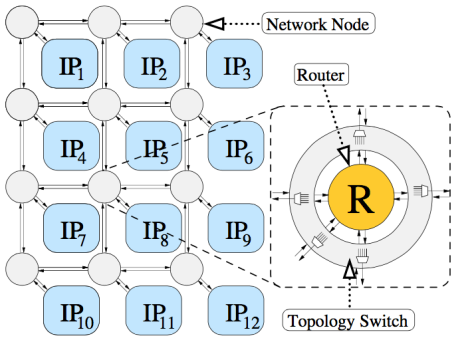


Figure 2.5: ReNoC Basic Components [32]

The figure 2.5 defines a basic summary of the ReNoC design, where the physical architecture of the ReNoC consists of nodes that are connected by links on a 2D topology, and each node consist of a regular router that is covered by a topology switch. Every switch is used to attach both links and routers to the logical topology that lies on top of the physical one. Connecting any two IPs, any two routers, or any IP and a router can form the long connections, while

using the clock gating to eliminate the dynamic energy consumption. Basically, topology switches can be connected directly to ports of routers or to links, and are they are designed for infrequent reconfiguration, as such once every time the chip is powered up, or at the start of a new application. By this way, area and energy efficiency are permitted to be implemented wisely, with respect to the design factors of the switch topology that is really an analogue to a switch-box in FPGAs. There is definitely an obvious separation between the switches and the routers implying no restriction on a certain router, which puts only one requirement of having the link width of a size that matches the ports of the routers [32].

2.7.3 Router Architecture

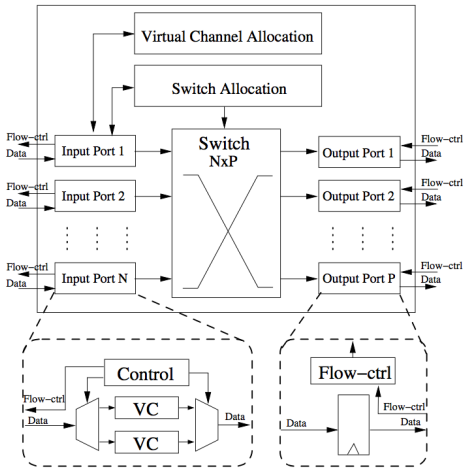


Figure 2.6: Overview of Router Architecture [32]

Figure 2.6 shows an overview of the router architecture used in this work. The router is regular source-router router that has two VCs, and each of its buffer can hold four flits. Each VC's buffer is implemented using a register file. Each dedicated packet has a header, followed by three body flits, where each flit is of size 34 bits. The flit has 32 bits for the header and data, while the other 2 bits are used to indicate the VC and the last flit in the packet. Other than the 32 bits, the link holds one bit that indicates the presence of a flit,

and also, another 2 bits that are used for flow control. Therefore, The total number of bits 37 bits.

2.7.4 Results and Discussion

Results acquired used about 25% of the routers, while others remained power-gated, which decreased the power leakage and idle power consumption. It was perceived that the area of the ReNoC created an overhead comparatively to the static mesh, where an increase of 10% was found. When ReNoC was configured for application-specific topologies, an improvement of 56% in power consumption was achieved. The paper discussed using routers with less number of ports in order to reduce energy consumption and area causing the topology to no longer resemble a two-dimensional mesh. The work was evaluated using low power routers and frequency of $100MHz$.

	Area (mm^2)			Power consumption (mW)					
Architecture	**Routers**	**Topology switches**	**Total**	**Routers**	**Topology switches**	**Links**	**Leakage Power Power**	**Idle Power**	**Total**
Static mesh	0.53	-	0.53	2.39	-	0.84	0.08	1.25	4.56
ReNoC mesh	0.53	0.05	0.58	2.39	0.12	0.84	0.08	1.25	4.69
ReNoC specific	0.53	0.05	0.58	0.65	0.09	0.84	0.03	0.41	2.02

Figure 2.7: ReNoC Simulation Results [32]

2.8 Scalable Hybrid Wireless NoC Architectures

According to [17], multicore platforms are evolving developments of the SoC design. NoC interconnection fabrics are the promising future solution for the to achieve the performance needs. The increasing demand for high speed and low power interconnects imposes the need to look beyond the conventional infrastructures. There are many different possibilities to substitute the conventional interconnections, and the wireless NoCs (WiNoC) are on top

on the list. This paper presents the design methodologies and requirements for developing scalable WiNoC architectures. Results have shown that the throughput and latency performance of WiNoCs beats its wired equivalents. Furthermore, energy efficiency improves by orders of magnitude. Several traffic models have been used to evaluate the performance of WiNoCs.

Normally, in conventional NoCs data packets travel through complex switches, which consume a large amount of power, and perhaps affect performance. Some papers discussed the introduction of VCs in order to improve performance. Moreover, performance is improved even better by introducing the ultralow latency on-chip global lines. While in other works, performance has been improved by inserting wide-range wired links in subsequence to small world graph theories. Even though all previous works showed significant performance, improvements has a limit due to the addition of extra links to the network. Further advancements will be considerably present if 3D integration circuits' technology is adopted and the combinations of the two technologies of NoCs and 3D ICs will contribute substantial improvements. Even though there are great benefit from the later mentioned technology, the thinning of wafers require highly interacted patterning. There is a lot of great work done to improve NoC performance, but we still suffer from many complications when implementing those new techniques [17].

The wireless on-chip interconnection fabrics were first exhibited in [14] to help in distributing clock signals. Some works have built on this technology and presented wireless transmissions of range 1 mm, this technology for a bit larger die size, will require multi-hops to communicate through its wireless channels. Authors of [17] were inspired by that work and proposed a long-range on-chip wireless communication links for better energy efficiency and lower latency.

2.8.1 WiNoC Topology and Links Insertion

The small-work phenomenon is an interesting theory that can perfectly benefit the WiNoC technology. Small-world brings shortcuts to the NoC topology, which can significantly improve the performance. The goal of the small-world methodology is to provide a greatly efficient for wired and wireless NoC-based SoCs. In this work, the system is divided into multiple clusters of next-door core, which are called subnets. Subnets are smaller networks with shorter paths that regular NoC paths, and they bridging around the system. In Figure 2.8, shown a subnet with a mesh topology, where NoC switches and links are present and cores are connected to a central hub via direct links. All hubs are connected through a network on another level to form a hierarchical network. The upper level carries the small-world theory characteristics.The authors of the paper proposed a hybrid wired/wireless architecture, where hubs are interconnected through both wireless and wired stations that allow data transmission and reception [17].

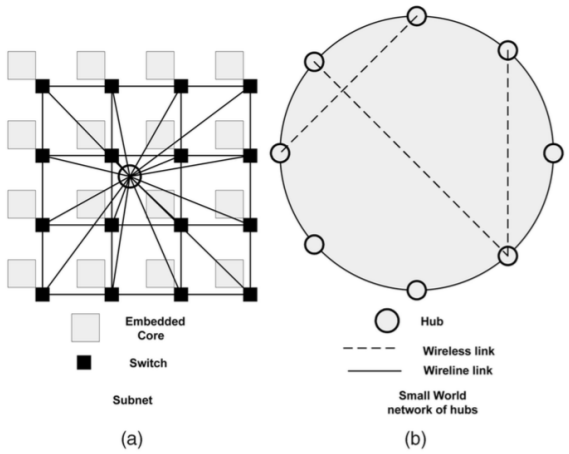


Figure 2.8: WiNoC Topology [17]

The paper discussed an optimized link insertion algorithm that starts with initializing the network, and then goes through optimization implemented via simulated annealing (SA) heuristic. The optimization is done only on the top-layer of the network, where this step is essential, as random initialization cannot produce an optimal topology. SA provides a simple

way to reach to a solution rather than the brute-force search technique. The first step to implement the SA heuristic is to define a metric, which is the average distance, measured in the total number of hopes between all sources to destination hubs. The distances are computed according to the routing algorithm that is set for the topology. Then, a new network is created with random wirings of the wireless links in the existing network in every iteration. A new metric is calculated and compared with the older network metric. The new network is always selected as the optimal solution not to get stuck in a local optimum [17].

2.8.2 Routing and Communication Protocol

The data routing in this proposed worked depends mainly on the subnets topology. Normally, the intersubnet data routes along the hubs through the shortest paths from source to destination subnets. There is a pre-routing block that the subnets are set with, to help search through the possible paths and determine the right path to take. In the work presented, there is a chance of taking paths that contain only wireless links. Potential paths are compared together, whether they were completely wireless or a mixture of wired/wireless links. When a data packet is required to route within an intersubnet, then computation is done once for the header at the source hub. Normally, when the wireless links are positioned as long-distance shortcuts, there shall be a comparison in length with the diameter of the ring. Therefore, the probability of having a path with multiple wireless links is exceedingly low. In order to get the best of trade-offs between network performance and routers complexity, only paths with a sole wireless link are put in consideration. Furthermore, if two distinct paths have the same number of hops, then the path that contain a wireless link will be selected, because this path will consume less energy. This path is determined in advance and calculations are done to make sure no deadlock or livelock would occur. Another approach is used to avoid evaluating one time to find the shortest path, where this mechanism implements a check at every node by computing and comparing the path distances.

2.8.3 Experimental Results

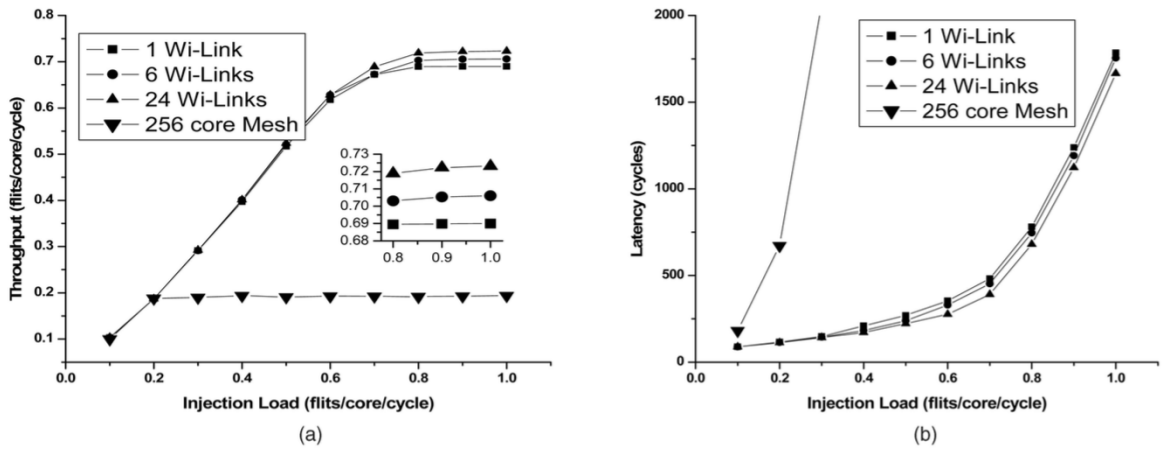


Figure 2.9: (a) Throughput and (b) latency of 256-core WiNoCs with different numbers of wireless links. [17]

The paper has analyzed the characteristics of their proposed WiNoC design and has done studies on the performance of the system. Three different sizes of networks have been tested, and those are 128 cores, 256 cores, and 512 cores on a die of size $20mm \times 20mm$. There have been multiple scenarios to analyze the results. In one scenario, a fixed number of cores for each subnet were used; while in another different sizes were considered for each subnet.

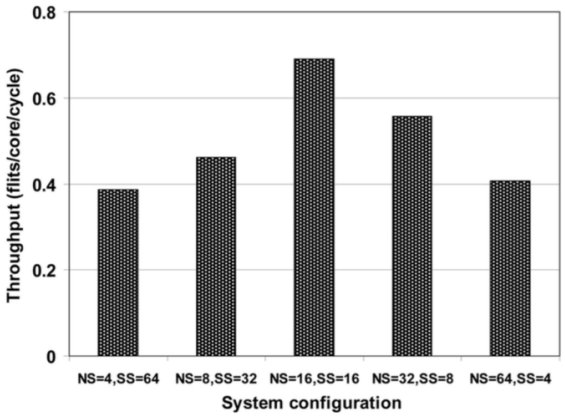


Figure 2.10: Throughput of 256-core WiNoC for various hierarchical [17]

Figure 2.9 shows the throughput and latency results as a function of the injection load for a system carrying 256 cores, where each 16 cores represent a district subnet, while Figure 2.10

show the saturation throughput for the different ways of splitting the cores into multiple subnets.The plot shows that the different alternative configurations reached worse saturation throughput, while if number of wireless links varied, the same behavior has been observed. When varying the number of channels in the wireless links, it means we can have different WiNoC configurations.

Figure 2.9 shows the energy dissipation of packets for a WiNoC with 24 wireless links on a mesh topology of 256 cores. The paper had discussions in about how WiNoC can improve the performance of a system and how beneficial it is in terms of reducing energy dissipation. It is noticeable that among the difference emerging NoC architectures, the hybrid NoC showed the lower packet energy dissipation, which can be tied to how WiNoC can reduce the average hop count compared to other NoCs. The paper have compared its architecture with other architectures such as [27], where wired shortcuts were used to implement the small-world network on a basic mesh topology. There was a huge gain of 0.75 flits/core/cycle that was achieved because of the hierarchical separation between the WiNoC and the wireless links layer. Energy dissipation was reduced by about 25%. This analysis has shown that the WiNoC significantly outperformed even current proposed NoCs. Furthermore, WiNoC is more energy efficient compared to other wired architectures.

Chapter 3

Problem Definition and Design Space Exploration

Trailing on the work presented by Omit Ogras in [27] that brings the notion of Small World Phenomenon into the application-specific NoC-based SoCs design, this newly proposed effort focus on extending the same idea in order to find the sub-optimal solution of placing a number of long-range links into general-purpose NoC-based SoCs. Because energy efficiency is a critical issue in the field of computer architecture, the improvement in energy efficiency we can get from adding extra links will definitely result in significant benefit to the NoC design [27]. We have essentially followed a similar track to [27] with focus on general purpose instead of application-specific SoCs. New routing and links insertion algorithms are used to fulfill the requirements and deliver the objective of this work. The basic idea states that we have links and routers consuming certain amount of energy in every unique NoC architecture. Vertical and horizontal links normally have equivalent size, and the length of a link determines the energy it consumes. While for the routers, the energy consumed depends on the number of the ports in each router. The program we have developed works on implying an incense of randomness to the NoC design in pursuance of the small world

theory [27], while upholding the basic architecture of a two-dimensional mesh topology. Along these lines, we keep the regularity of the standard topology, and profit the energy efficiency. The genetic algorithms approach is employed providing a strong backbone for the search procedure of the sub-optimal solution.

3.1 Basic Assumptions

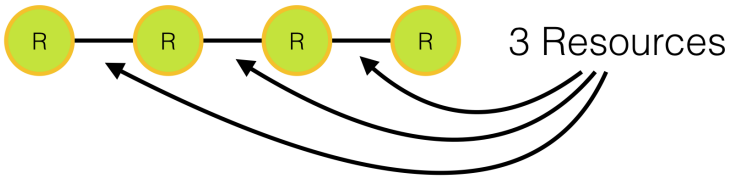


Figure 3.1: Resources Legend

In this work, the term resource is defined, where one resource is equivalent to the size of one short link connecting two routers together [27]. In Figure 3.1, a sub-network is extracted from a $4x4$ two-dimensional mesh topology. The figure shows four routers connected by three links, where the three links define the number of resources used to form the path connecting the routers together. Only vertical and horizontal links can be added to the network, therefore, the maximum number of resources for each link is equivalent to the distance between two adjacent sides of the network in terms of hop links. If we assume that we have an mxn mesh NoC, then the maximum size of the resources required to connect the sides of the network is $n-1$ or $m-1$ (See Figure 3.2). The minimum size of the long-range link obviously has to be equivalent to two resources, as one resource is defined to be the regular mesh connection of two routers. Furthermore, we do not allow adding more than one long-range link per port to scale down the overhead of extra links ports.

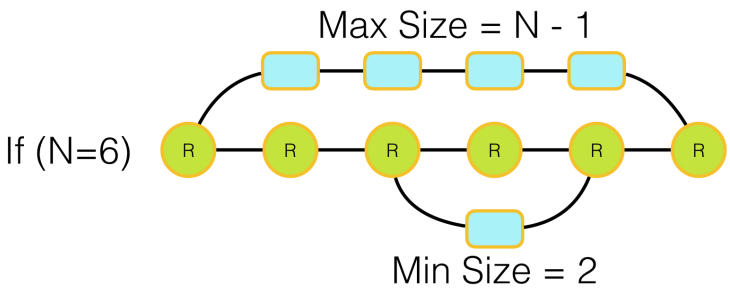


Figure 3.2: Long-Range Link Size Example

3.2 NoC Construction

Object Oriented Programming (OOP) languages conceal wide-ranging abilities to model and illustrate the real world. OOPs allow creating classes of objects that can easily define the behaviors of a model. Those objects can be used in multiple programs allowing reusability of the code. The modularity of OOP defines each object as a unique entity of itself, making it decoupled from other parts of the system. For the great benefits of OOPs, we selected Java programming language to implement this work.

The program developed starts initially by setting the size of the NoC topology (e.g. 8×8 mesh network) and the costs or the different components. A class NoC is defined to implement the functions required to connect and manage the elements of a NoC. Once the program runs, it creates the class NoC and initializes the objects for every element in a NoC. The objects can be tiles, links, or routers, which describe the NoCs main constituents. Each of those objects outlines a unique module with distinctive values and costs representing its location, power consumption, connections and/or components. For every object NoC, $n \times m$ $Tiles$ are created as shown in Algorithm 1. Each $Tile$ is assigned each to its (x,y) location in a two dimensional array. Afterwards, the links can be connected to the tiles forming a 2D mesh according the defined Algorithm 2.

Algorithm 1 Create NoC Tiles

```
1: procedure CreateTiles(tilesMatrix)
2:     TileID ← 0
3:     temp ← null
4:     for i = 0 to i > numberOfRows do
5:         for j = 0 to j > numberOfColumns do
6:             temp ← newTile(tileID)
7:             temp.setxLocation(i)
8:             temp.setyLocation(j)
9:             tilesMatrix[j][k] = temp
10:            tileID ← tileID + 1
11:        end for
12:    end for
13: end procedure
```

3.2.1 Links and Connections

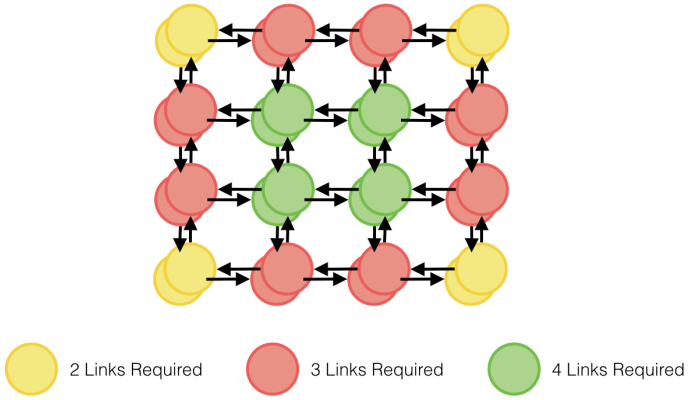
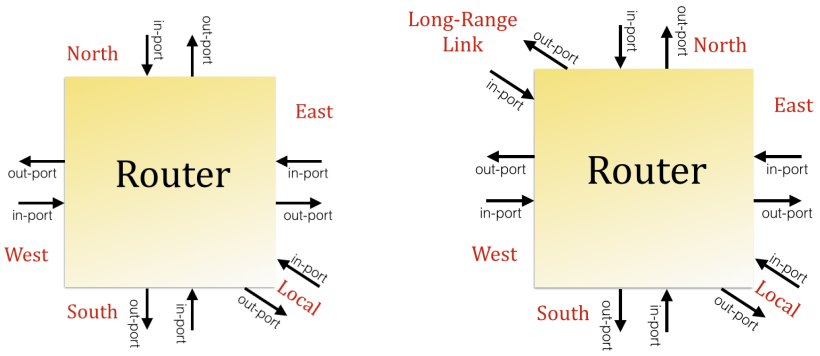


Figure 3.3: Number of links required to connect the different located tiles on a NoC

The class *Link* terms eight types of Link objects representing four directions of connections: right, left, up, down, that are used to define the two types of connections (regular and long-range links). The first four links are equivalent to each other in all terms as they are descendants of the same class, but they differ with the direction of *Link*. The cost of a long-range link include the cost of all hop links and the repeaters connecting them. In the program, every tile has two input and output ports and each port connect input and output links. If a tile is central, then it has eight links connected to it; four input links, and



(a) Conventional Router (b) Extra Port for Long Links

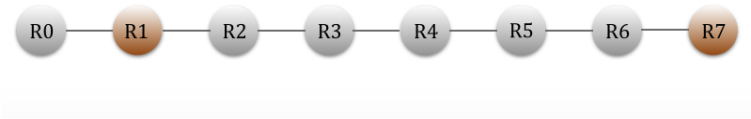
four output links. In this work, local ports are not considered, as they will not be used in calculating the energy efficiency costs or simulations. Links can be connected to the NoC design as explained in Algorithm 2. The function *connectLinks*() loops through the X and Y coordinates and makes sure to connect the required links for each tile on the two-dimensional mesh. Normally, the corner tiles require only two directions to send/receive packets to the neighboring tiles, while the center tiles needs connections in the four directions. Depending on the location of the tile, links will be set for each of the $m \times n$ tiles (See Figure 3.3).

3.2.2 Routers and Ports

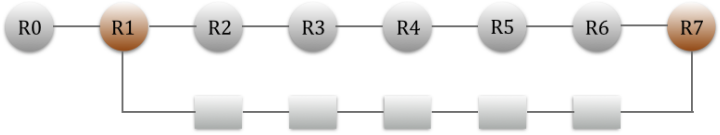
A typical NoC consist mainly of three components: the processing elements, the routers, and the network interfaces, where routers and network interfaces are responsible for routing the packets through the network. Packets move hop-by-hop when traversed from source to destination. Generally, once a packet is received, then it is stored in an input-buffer waiting for the router's control logic to decide the next destination. Packets keep traversing through the network from one router to the other until they reach the designated destination [12]. Figure 3.4a shows the basic router architecture, where every router can possibly have input and output port to (East, West, North, South) and a Local input and output ports, while in Figure 3.4b, an extra port is added to the router design in order to satisfy the added extra

Algorithm 2 Connect Links to Tiles

```
1: procedure connectLinks(tilesMatrix)
2:     Link tempLink
3:     for k = 0 to k < NumberOfColumns do
4:         for j = 0 to j < NumberOfRows do
5:             if j ≠ NumberOfColumns − 1 then
6:                 tempLink ← newLink()
7:                 if tilesMatrix[j][k].getRightLink() ← null then
8:                     tempLink.tile1 = tilesMatrix[j][k]
9:                     tempLink.tile2 = tilesMatrix[j + 1][k]
10:                    tempLink.setLinkCost(xCost)
11:                    tilesMatrix[j][k].setRightLink(tempLink)
12:                end if
13:            end if
14:            if k ≠ NumberOfRows − 1 then
15:                tempLink = newLink()
16:                if tilesMatrix[j][k].getLowerLink() = null then
17:                    ...
18:                end if
19:            end if
20:            if j ≠ 0 then
21:                tempLink = newLink()
22:                if tilesMatrix[j][k].getLeftLink() ← null then
23:                    ...
24:                end if
25:            end if
26:            if k ≠ 0 then
27:                tempLink = newLink()
28:                if tilesMatrix[j][k].getUpperLink() == null then
29:                    ...
30:                end if
31:            end if
32:        end for
33:    end for
34:    return tilesMatrix
35: end procedure
```



(a) Regular NoC Interconnections



(b) Extra Long-Range Links Connected Via Repeaters

Figure 3.5: To implement the idea of inserting ling-range links to NoC topologies, we need to use repeaters to avoid delays caused by long wiring

long-range link needs.

In this work, a distinct router is assigned to each tile. Every instance of the class *Router* has its own unique identity generated along with it once its instantiated to distinguish between the different objects in the network. Since the main consideration is energy efficiency, the cost of each router is set depending on the number of ports it carries. The costs are predefined and the suitable cost is assigned to the routers once their objects are created. In Figure 3.3, it is shown that the corner routers normally need ports for only two directions, central routers require ports to connect to the four directions, while the side routers will need only three. An extra port has to be added for some routers with the extra long-range link Since only once extra link is allowed per router, the maximum number o ports allowed for a router is five.

3.3 Repeaters

Since proposing long interconnects into SoCs obviously result in delays, adding repeaters in the appropriate locations is one of the most common techniques to overcome such com-

plication [19]. Typically, repeaters consume a lot less energy than routers, and by placing repeaters in the same positions of routers to connect two remotely located nodes as a means to achieve our goal. Figure 3.5a shows an example of a row from a regular mesh topology including normal NoC interconnects, and in order to travel from Source (R1) to Destination (R7), packets are required to go through the route (R1, R2, R3, R4, R5, R6, R7) making the system consume energy for passing through all those routers from R1 to R7, plus, the energy required by the links to transport through. On the other hand, in Figure 3.5b we can see that the packets requested to travel from the same Source (R1) to the Destination (R7) can use the extra long-range link, where instead of going through the path (R1, R2, R3, R4, R5, R6, R7), packets can directly proceed from $Router1$ to $Router7$ using the low-power repeaters. Even though the routers connecting the long-range link will have an extra port that will increase the energy consumed by a router comparatively to the original mesh design, this issue can be overcome. The longer the links and the more repeaters, the less energy is consumed by that path comparing to the regular mesh. Traversing through the path with the repeaters will definitely improve the energy efficiency.

3.4 Long-Range Links Placement

In order to formulate a methodology to select and place the links a two-dimensional mesh topology, calculating the total number of the links we possible add to the network is required. The value of the possible links is obtained to define the size of two-dimensional array, where the key ID for every column is within the range of 0 and the value of the maximum possible links. Each column has its unique ID and the information associated with every link in the network is assigned to one column. Such information include the coordinates (x, y) for the node location, the cost of the link, and whether the link is available in the network or not.

3.4.1 Calculate Total Number of Possible Links

In every mesh topology, there is a p number of link choices that can be present in the network. For the basic mesh topology, the number of available links in a network known and the available links are fixed. Allowing the insertion of extra links create even more options of links to be present in the network. In this work, the extra links are restricted to be present only horizontally and vertically in a mesh topology and only one extra link can be attached per router. Such assumption was made to allow only slight modifications to the basic network, while keeping the regularity of the mesh topology. The number of possible links p is calculated as shown in Listing 3. For a given $n \times m$ dimensions, there are np possible links for n dimension and mp possible links for m dimension, where $np + mp = p$. The mathematical rule behind the equation implemented in the code is according to the Combination formula:

$$\binom{n}{k} = \frac{n!}{k!(n-k)!} \tag{3.1}$$

Where n is the number of nodes in a row/column, and r is the number of selection items. Further information was added to the formula to fulfill the goal of it. Since the above mentioned formula calculate only the links for every row. We needed to multiply it by m rows, which will eventually give the total number of links possible in every row. Finding the the number of possible links for every column is considered too. In this work, it is assumed that the size of a column might be different than a size of a row. In this case, the same equation is implemented for both rows and columns, and the outputs are summed up to find

out the total number of possible links 3.2.

$$m\binom{m}{k} + n\binom{n}{k} = \frac{m!}{k!(m-k)!} + \frac{n!}{k!(n-k)!} \tag{3.2}$$

The same equation was transformed to calculate the total possible allowed links for a NoC in the java code implementation (Equation 3.3).

$$\begin{aligned} Totalnumberoflinks = ((m-1) + (m-2) + ... + (m-(m-1))) * m + \\ ((n-1) + (n-2) + ... + (n-(n-1))) * n \end{aligned} \tag{3.3}$$

In Figure 3.6 is shown all the possible links that could be present in a 5×5 mesh topology according to the basic assumptions, which restricted any diagonal connections. Referring to Listing 3, the calculations starts with $row = 0$, and counts only the number of links connecting $PE0$ to every other node on the same row. Next, the code turns to the next node $PE1$ and keep counting the links connecting it to the other nodes on the same row, while making sure not to include the links counted previously for $PE0$. The permutations are not considered, and only combinations are counted, which means that a link connecting $PE0$ to $PE1$ is equivalent to the link connecting $PE1$ to $PE0$ and shall be counted only once. The counting continues until the penultimate node is reached, where it will only have one connection to count and the loop will stop. All the other rows will follow the same technique to count the number of links, therefore, links are only counted once for one row and then multiplied by the number of rows m to find the total number of possible links in the rows. Because it is assumed that there is a possibility that the number of rows might not be equivalent to the number of columns, the exact procedure is implemented to count

the number of possible links for the columns. At the end, both results are added up to come out with the total number of possible links for a NoC. By implementing the Equation 3.3 for the example shown in the figure, the following results are obtained:

$$\begin{aligned} Totalnumberoflinks &= ((5-1)+(5-2)+(5-3)+(5-(5-1)))*5+ \\ &\quad ((5-1)+(5-2)+(5-3)+(5-(5-1)))*5 \\ &= ((4+3+2+1)*5)+((4+3+2+1)*5) \\ &= 100\ links \end{aligned} \tag{3.4}$$

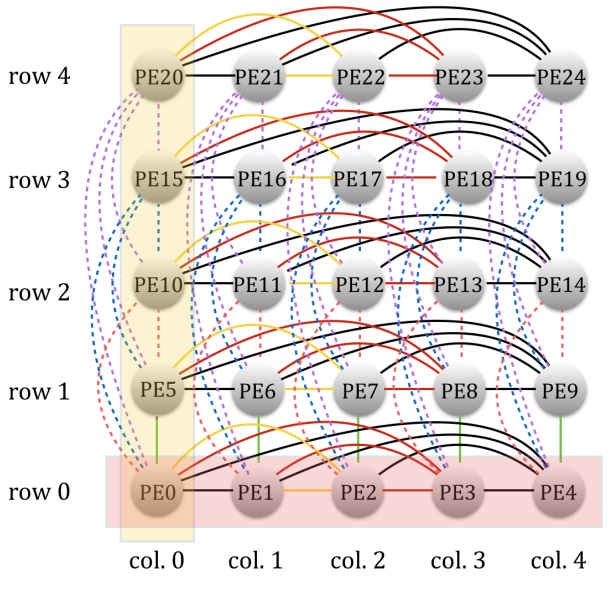


Figure 3.6: The figure shows all the possible links that could be added to the 2D mesh with restrictions to add any diagonal links

3.4.2 Calculating Costs of NoC Links

After obtaining the total number of possible links in a mesh network, a unique ID is set for each link to associate all information related to the link with its defined ID. A two dimensional array is defined and the its first row carry the key IDs ranging from 0 to the

total number of possible links. Each column represents the information of one possible link in the network. The information related to the link can be: the coordinates of the tiles the link is connecting, the cost of the link, and whether the link is available on the network or not.

Link ID	0	1	2	3	4	5	6	7	8	9	10	11	.....	42	43	44	45	46	47
x 1	0	0	0	1	1	2	0	0	0	1	1	2	.....	0	0	0	1	1	2
y1	0	0	0	0	0	0	1	1	1	1	1	1	.....	3	3	3	3	3	3
x2	1	2	3	2	3	3	1	2	3	2	3	3	.....	1	2	3	2	3	3
y2	0	0	0	0	0	0	1	1	1	1	1	1	.....	3	3	3	3	3	3
Cost	..	..	..	..	..	..	..	..	..	..	..	..	.....	..	..	..	..	..	..
Availability	..	..	..	..	..	..	..	..	..	..	..	..	.....	..	..	..	..	..	..

Figure 3.7: *linksArray* is a 2D array that holds all information associated with a link. The figure shows an example of information available for a 4×4 mesh topology

3.4.3 Link Placement

Placing long-range links has to be done wisely in order to truly enhance the energy efficiency of a NoC. Finding the best possible design is considered one of the NP-Hard optimization problems, and solutions to such problems have not yet been found. Fortuitously, there are several techniques that may help narrow the search space and provide a satisfactory sub-optimal solution. Evolutionary algorithms are useful in finding reasonable solutions to the problem in a sensible amount of time. Genetic algorithm is one of the descendants of the evolutionary algorithms. It is a heuristic based technique that mimics certain genetic behaviors to bring forth a sub-optimal solution efficiently and in a considerable time. A Genetic Algorithm is used in this work to generate initially a set of architectures (individuals), and carry it through the steps of genetic algorithm until a satisfying result is obtained. Since genetic algorithms are used to solve the problem, placing extra long-range links randomly is the first step in defining the basis for the genetic algorithm.

A code is developed to primarily receive a number of resources R, where R is less than the

allowed total number of possible links. There is no specific rule stating a definite amount of resources R. By doing some calculations, we can find logical and reasonable number of resources to set as an input for the network. As discussed previously in Subsection 3.4.1 in the total possible links example for a 5×5 dimensions network, it was found that the total number of possible links is 100. For those 100 links, the total number of resources $T_{resources}$ is calculated according to the following equation:

$$\begin{aligned} T_{resources} &= (((m-1)*(m-(m-1))) + ((m-2)*(m-(m-2))) + ...+ \\ &\quad ((m-(m-2))*(m-2)) + ((m-(m-1))*(m-1)))*m \\ &= (((n-1)*(n-(n-1))) + ((n-2)*(n-(n-2))) + ...+ \\ &\quad ((n-(n-2))*(m-2)) + ((n-(n-1))*(n-1)))*n \end{aligned} \tag{3.5}$$

$$\begin{aligned} T_{resources} &= ((((5-1)*(5-(5-1))) + ((5-2)*(5-(5-3)))+ \\ &\quad (5-(5-2))*(5-2)) + ((5-(5-1))*(5-1)))*5+ \\ &\quad ((((5-1)*(5-(n-1))) + ((5-2)*(5-(5-3)))+ \\ &\quad (5-(5-2))*(5-2)) + ((5-(5-1))*(5-1)))*5 \\ &= ((4*1)+(3*2)+(2*3)+(1*4))*5 + ((4*1)+(3*2)+(2*3)+(1*4))*5 \\ &= 200 \ resources \end{aligned} \tag{3.6}$$

According to the assumptions, it is expected to preserve the regularity of the mesh network. This, results in having even less number of link possibilities, where the regular mesh links are not counted and only the long-range links are considered, resulting in T_{LRLs} equivalent

to 60 link possibilities:

$$\begin{aligned} T_{possibleLRLs} = {} & (m-2)+(m-3)+.....+(m-(m-1))*m+ \\ & (n-2)+(n-3)+.....+(n-(n-1))*n \end{aligned} \tag{3.7}$$

$$\begin{aligned} T_{possibleLRL} = {} & ((5-2)+(5-3)+(5-(5-1)))*5+ \\ & ((5-2)+(5-3)+(5-(5-1)))*5 \\ & = (3+2+1)*5+(3+2+1)*5 \\ & = 60 \ links \end{aligned} \tag{3.8}$$

Generally, those 60 possibilities of extra long-range links can have total resources $T_{pLRL_{resources}}$ that is about 4/5 of the total network resources $T_{resources}$. Since the basic mesh topology links are normally shorter in size, they tend to cost less resources than the long-range links:

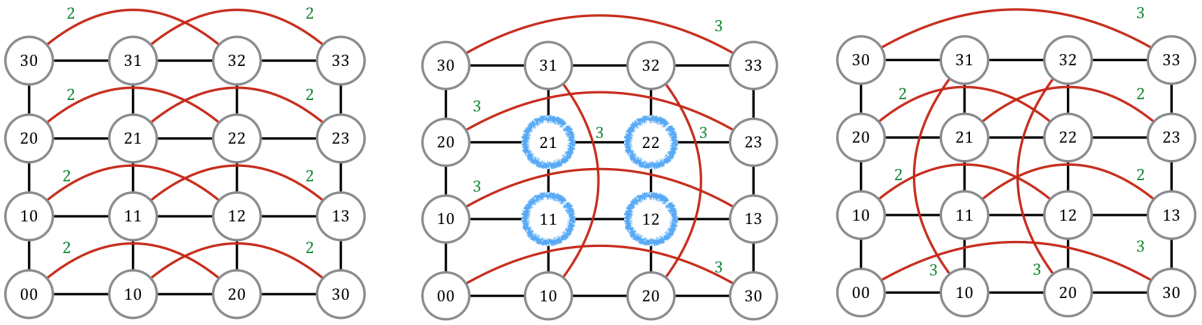
$$\begin{aligned} T_{pLRLs_{resources}} = {} & (((m-(m-1))*(m-1))+((m-(m-2))*(m-2))+.....+ \\ & ((m-2)*(m-(m-2))))*5+ \\ & (((n-(n-1))*(n-1))+((n-(n-2))*(n-2))+.....+ \\ & ((n-2)*(n-(n-2))))*5 \end{aligned} \tag{3.9}$$

$$
\begin{aligned}
T_{pLRLs_{resources}} = & (((5-(5-1))*(5-1))+((5-(5-2))*(5-2))+ \\
& ((5-2)*(5-(5-2))))*5+ \\
& (((5-(5-1))*(5-1))+((5-(5-2))*(5-2))+ \\
& ((5-2)*(5-(5-2))))*5 \\
& = ((1*4)+(2*3)+(3*2))*5+((1*4)+(2*3)+(3*2))*5 \\
& = (4+6+6)*5+(4+6+6)*5 \\
& = 160 \ resources
\end{aligned}
\tag{3.10}
$$

A constraint stating that a router cannot connect to more than one extra link per router aside from the basic constant mesh connections was mentioned earlier. Accordingly, the total resources available for adding the long-range links are even less than what has been calculated previously. The constraint discussed earlier sets a limit to the number of permitted resources. The number of links and the number of resources has been analyzed and compared to each other in order to define a logical connection between the number of links and the number of resources. Figure 3.8 shows three different ways of adding extra links to the network.

Allowing the addition of the minimum size of a link that is of size 2 resources as described in the example shown in Figure 3.8a helps obtaining the maximum number of extra links. According to Table 3.1 for a 4×4 mesh network, there is a total of 16 nodes, and in order to concentrate the network with the minimum size of extra link, there is $16/2 = 8$ connections with a total LRL resources of 16. Similarly, for the 5×5, there are 25 nodes and a maximum number of links $= 25/2 = 12$ making about 24 resources.

In the second case 3.8b, the maximum size of long-range links is added to the mesh network, the results shown in Table 3.1 are obtained. A greater number of resources is obtained than what is shown in Table 3.8a, allowing better utilization of the energy consumption when



(a) Shorter Links (b) Longer Links (c) Multi-Size Links

Figure 3.8: 4x4 Mesh with Different Links

Dimensions	Min Possible Extra Links	Max Possible Extra Links	Max Resources	Nodes with No Links
4x4	0	8	~16	0
5x5	0	12	~24	1
6x6	0	18	~36	0
7x7	0	24	~48	1
8x8	0	32	~64	0

Dimensions	Max Possible Extra Links	Max Possible Extra Links	Max Resources	Nodes with No Links
4x4	0	6	~18	4
5x5	0	12	~44	1
6x6	0	16	~68	4
7x7	0	24	~116	1
8x8	0	30	~166	4

Table 3.1: Minimum and Maximum Resources Table

passing through longer paths. Even if better utilization is achieved, it is hard to tell if this is the best possible method for implementing the idea. In industry, adding more wiring might not be feasible, therefore, finding a way to add less number of links to give it a chance for industry implementation while satisfying the goal of energy optimization is considered. Figure 3.8 allows having more links, which alternatively means, more paths to go through and more resources allowed. By combining the two cases of having more links and more resources, better results are achieved, as the longer the links and the more paths there is, the higher the chance of taking the new paths and greater optimization is gained. Genetic Algorithms are used to implement this work and is discussed in details in Section 3.7.3.

Algorithm 3 Calculate Total Possible Long-Range Links Algorithm

```
1: procedure CALCULATE(n, m)          ▷ n=number of rows, m=number of columns
2:     p ← 0                                                ▷ Total possible links
3:     np ← 0                                          ▷ Total possible LRLs on the X-Axis
4:     mp ← 0                                          ▷ Total possible LRLS on the Y-Axis
5:     temp ← n
6:     while temp ≠ 0 do          ▷ Calculate the number of possible links for one row
7:         np = np + (temp − 1)
8:         temp = temp − 1
9:     end while
10:    np = np ∗ n                                          ▷ Multiply by number of rows
11:    temp ← m
12:    while temp ≠ 0 do       ▷ Calculate the number of possible links for one column
13:        mp = mp + (temp − 1)
14:        temp = temp − 1
15:    end while
16:    mp = mp ∗ m                                       ▷ Multiply by number of columns
17:    p = np + mp                                                  ▷ Sum results
18: end procedure
```

3.5 Link Selection Algorithm

Every link has unique identity whether it was a short-range link connecting two adjacent nodes, or a long-range link connecting two remotely situated nodes. Initially, the link selection algorithm creates the nodes and connects them to form the regular mesh architecture. Afterwards, A long-range *link* is selected, making sure that the selected *link* has not previously been added, or the source/destination have not a long-range link connected to them, because we set a constraint that acknowledges connecting only a single long-range link to a specific router. In this approach, a suitable length for the allowed wiring in terms of number of hops is specified, where one hop is equivalent to one short-range link connecting neighboring nodes. The algorithm tries to add links of varying sizes ($link.Size$) until a total links size equivalent to the specified allowed length ($allowedLinks$) is connected (See Listing 4). There might be cases where some extra links are selected and added to the architecture, and at certain point, the algorithm tries many times to select other links and fails to add them

to the network. This failure occurs when either the link had been selected previously, or one of the nodes it is connecting has already another extra link attached to it. In such cases, after a certain number of failed trials to add new links, all the added long-range links are removed, and procedure is restarted to insert a new combination of links.

Algorithm 4 Link Selection Algorithm

```
1: procedure SELECTION(allowedLinks, individual)        ▷ Total allowed wiring for long-range links
2:     link ← RandomSelect()                            ▷ Select a random long-range link
3:     while allowedLinks ≠ 0 do
4:         if link ∉ individual.links then
5:             if individual.getLink(link).getSource().getLRL() = null then
6:                 if individual.getLink(link).getDestination().getLRL() = null then
7:                     individual.Connect(link)
8:                     allowedLinks = −link.Size
9:                 end if
10:            end if
11:        end if
12:    end while
   return individual
13: end procedure
```

3.6 Deadlock-free Routing with Extra Long-Range Links

Once the basic NoC architecture is configured and the long-range links are successfully added to the mesh NoC, selecting the best possible path to travel through from source to destination is the next step. There are several factors that need to be considered to determine the appropriate conceivable path. One of the factors is the knowledge about the routing algorithms and the power estimates for links, repeaters, and routers. There are many routing algorithms that can be candidates for implementation in this work. XY routing algorithm is selected. XY routing algorithm is a deterministic routing algorithm is selected to be the basic routing algorithm because of its simplicity. Generally, in an XY routing algorithm, flits are routed from their source through the X-axis until they get

to the Y coordinates of their destination. Then packets move through the Y-axis towards destination. The XY routing algorithm is definitely a deadlock free routing algorithm, as it follows the design rules of the turn model discussed in [18], which ultimately result in deadlock free routing. Basically, the XY routing algorithm is defined for use in the paths that move through routers with no long-range links, while another technique it used to hop through routers connecting extra long-range link.

Since this work focuses mainly on energy efficient routes, making great use of the extra long-range links is seriously considered. In many cases, hopping through the extra long-range links will result in better routes in terms of energy consumption, while in some cases it would be better to go through the regular mesh links. Some routes that take the extra long-range links might require switching completely from hopping to the west to hopping to the east, or from south to north or vice versa. Such instant switch is considered a U-turn, which will definitely cause a deadlock in the network. For such routes, VCs are implemented to handle those turns and allow routes to switch from one VC to the other. VCs are implemented in Torus Topologies to avoid deadlocks, and it has been proved in [6] that the long links in the Torus topology are deadlock-free when using VCs.

The network is configured with two VCs in each axis. Normally, when using the XY routing algorithm, packets will always move towards the destination, but when there are long-range links in the network, it is not always the case. Moving through some long-range links that are located to the opposite side of the destination may result in a more energy efficient route. Assuming that packets are routed via XY routing algorithm and have to move from east to west to reach the destination. A node located to the east of the source node has a long-range link that provides a shorter path to the west and is more energy efficient than the regular mesh path. Then, it is more effective to move through the long-range link path, which, in the normal cases, can cause deadlock. Since VCs are implemented, deadlock issues are resolved, as once a U-turn is present, the packets switch from $VC1$ to $VC2$ or vice versa.

3.7 Calculating Path Cost

Now that the deadlock free routing algorithm is defined and the links, routers, and repeaters costs are available, calculating the cost of traveling from source to destination can be determined. It is required first to find all the possible paths connecting two nodes together. Implementing the steps of the basic XY routing algorithm is the first step to find all the available paths between two nodes. The first step in the XY routing algorithm is to hop through the X-axis, and the second step is to hop through the Y-axis towards the destination. The intersection of those two paths creates the complete route from source to destination. Figure 3.9 shows an example of the X-axis and Y-axis needed to find the path from Source $P38$ to Destination $PE9$. The row and column highlighted in *yellow* show the axises to traverse through in order to reach the designated destination. Usually, in the XY-routing algorithm, the direction of the packet movement from source to destination is fixed, because XY is a deterministic routing algorithm. Packets will always hop through one X-axis and one Y-axis to reach their destination. According to this, whenever finding paths connecting two nodes is requested, the rows and columns needed for this operation are pre-defined. Therefore, there is no necessity to consider other rows and columns within the network, and only paths present in those pre-defined axises are considered.

3.7.1 Least-Energy-Cost Path Using Dijkstra Algorithm

By using to Dijkstra [13], the shortest path between a node to a another on a graph can be determined. In this work, the $n \times m$ nodes of the two-dimensional mesh topology are considered to be the graph that is used by the Dijkstra algorithm to find the shortest from one node to the other. The distance between two nodes is defined by the sum of the cost of the links, routers, and repeaters needed for traversing packets within a path. Generally, considering the whole mesh topology for finding paths from source to destination is not a

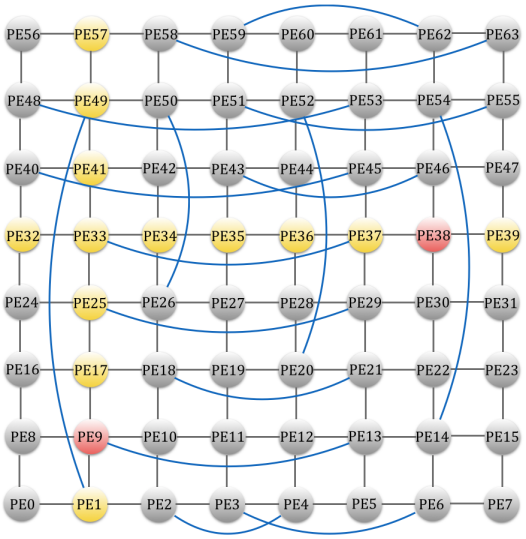
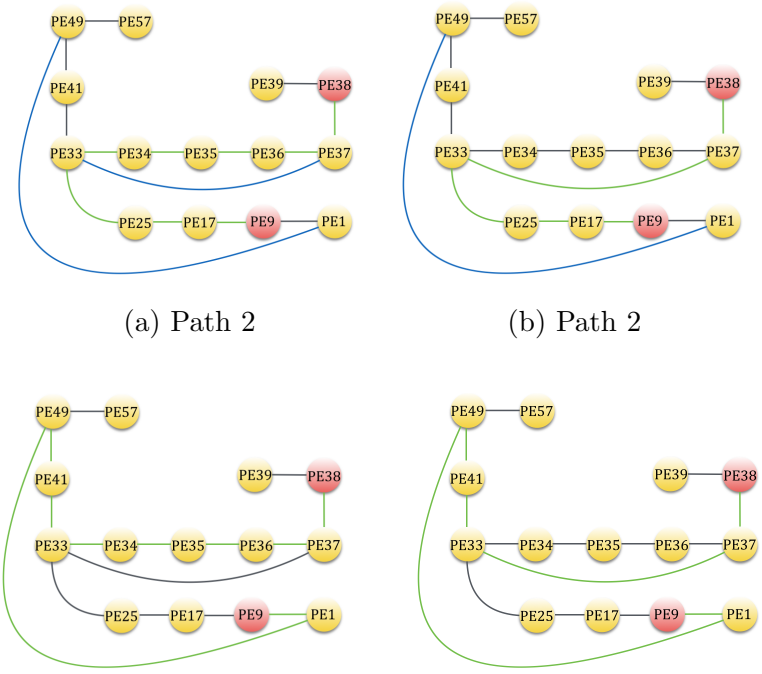


Figure 3.9: Selecting a Sub-Graph

logical solution to the problem.

Since XY routing algorithm is employed, then only the row and the column required for creating the paths from source to destination are taken into consideration. The choices for the targeted path are dramatically narrowed as only paths that are within the X-Axis and/or Y-Axis carrying the source and destination are considered as shown in 3.9. Typically, when both source and destination are located at the same axis, considering both X-Axis and Y-Axis for the paths is not required. Accordingly, the graph will implement the Dijkstra algorithm is constructed from the row and/or the column enclosing the source and destination nodes. Now, the graph holds all possible available paths for the row and/or the column leading packets from the source S to the destination D (See Figure 3.9).

Referring to Figure 3.10, the green lines on the four Figures 3.10a, 3.10b, 3.10c, 3.10d represent the different available paths directing packets from source $PE38$ to destination 9 (See Figure 3.9). As previously discussed, going through any of these path will hold a certain cost depending on the number of links and repeaters, and costs of different routers. By implementing Dijkstra algorithm, the shortest path costing least energy is selected as the ideal path to send packets through.



(a) Path 2 (b) Path 2

(c) Path 3 (d) Path 4

Figure 3.10: Different Paths from Source to Destination

Figure 3.11 shows some realistic costs that is used in the simulations for the 8×8 two-dimensional mesh. The data used on the figure are taken from Table 4.3. When function *Dijkestra* is called, the extracted sub-graph is given as an input along with the costs its nodes and links are carrying. According to those costs, the Dijkstra algorithm will traverse through the graph and find which is the path that will cost less energy.

The data in Table 3.2 shows every possible path from node $PE38$ to $PE9$. The first column shows arrays of the four available paths to traverse the packets through, as described in Figure 3.11. The other three columns Links, Routers and Repeaters represent the sum of the costs of all links, routers, or repeaters for that specific path. The last column gives the total sum of energy used by a path. In this example, the path shown in red color in the figure defines the shortest path in terms of energy consumption. When going through the path with the extra link, packets traveled through repeaters costing less energy, thus, energy used is reduced comparatively to other paths. Even through there were other paths that

used two extra links, those paths did not show better energy efficiency. More U-turns and longer distances reduced the energy efficiency of those paths.

The implementation of Dijkestra algorithm will be used in the Genetic Algorithm described in Section 3.7.2 to find the fitness for every possible solution to the defined problem.

Path	Links	Routers	Repeaters	Sum
38,37,33,41,49,1,9	8220	6716	1200	16136
38,37,33,25,17,9	4384	6222	450	11056
38,37,36,35,34,33,41,49,1,9	7672	9284	750	17707
38,37,36,35,34,33,5,17,9	4384	8790	0	13174

Table 3.2: Dijkstra Paths Costs for the 8x8 Mesh Example

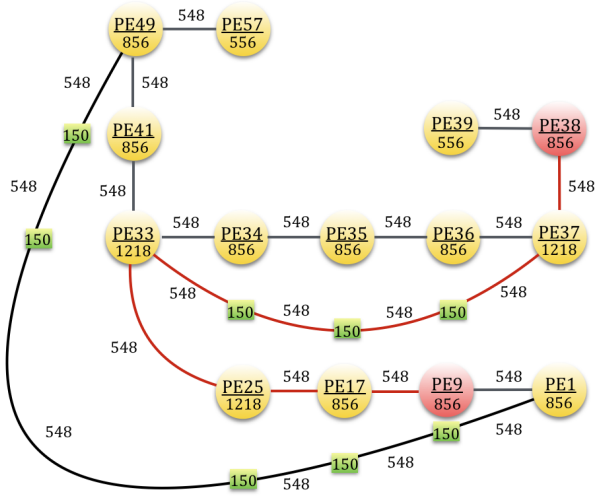


Figure 3.11: Paths Costs

3.7.2 Genetic Algorithms Operators

In accordance to [3], there are certain operators that have to be defined before implementing a Genetic Algorithm. A genetic algorithm searches through a space of chromosomes, which, in this case, are the different designs of 2D architectures with the extra links. Every genetic algorithm has a set of elements consisting of: populations of chromosomes, selection set of chromosomes, crossover to produce new offspring, and a random mutation for then newly

generated offspring. Chromosomes usually take the form of a string binary numbers, and the search for new solutions happens through changing the binary representation of chromosomes. Every different combination of bits, represent a unique solution to the problem. The genetic algorithms use what is called a fitness function, in which, each chromosome has a score for its fitness. The fitness defines how well a chromosome can solve the problem.

3.7.3 Sub-Optimal Solution via Genetic Algorithm

Conferring to [23], the formulation of a general genetic algorithm problem has to be clearly defined, and a convenient solution can be discovered using a bit-string representations for the candidate solutions. Accordingly, for a given two-dimensional NoC mesh topology of size $n \times m$, it is requisite to find the best possible placements of extra long-range links as a means to reduce energy consumption, while satisfying wiring and links constraints. There are mainly five steps to implement the genetic algorithm for a well-defined problem and can be implemented to our problem as follows (Refer to Figure 3.12):

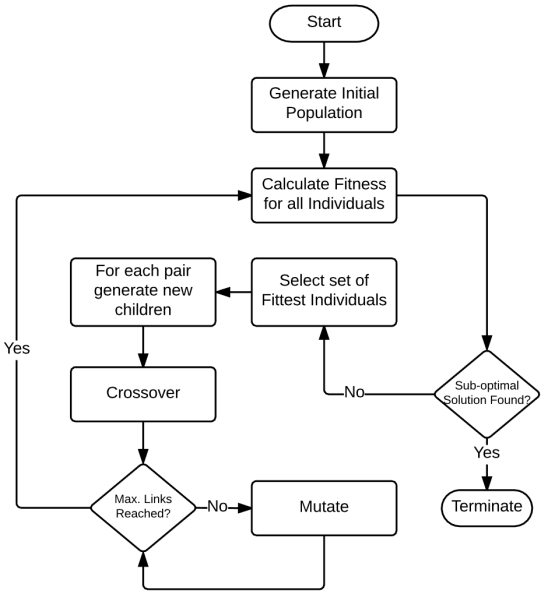
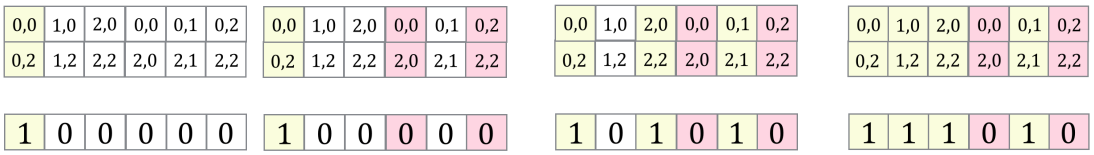


Figure 3.12: Steps of Implementing the Genetic Algorithm

3.7.4 Generate Initial Population



(a) Connecting (0,0) and (0,2)

(b) Links colored in red, cannot be added used

(c) Connecting (0,0) to (0,2), and (0,1) to (2,1)

(d) Connecting (1,0) and (1,2)

Figure 3.13: Links Selection Process

Primarily, a population *pop* holds a set of n initial candidate solutions, and each individual i is a NoC with extra long-range-links. Each individual is represented by a string of n 1-bit chromosomes. Each bit in the string symbolize a present/absent long-range link in the architecture. For example, Figure 3.13 shows a step-by-step links selection and insertion to the architecture. When creating each individual, its extra links are selected randomly. Once a link is selected, other possible links using the same nodes of the selected links will be omitted from future link selections, and that is referred to the constraint set in the basic assumptions to block adding more that one long-range link per router (See Figure 3.13b).

3.7.5 Calculating Fitness and Fitness Function Formulation

For every chromosome, calculating the fitness function $f(x)$ is required to define how fit an individual is in the terms we define. Thereupon, depending on how much energy an architecture consumes, the fitness function $f(x)$ is defined. The Dijkstra algorithm discussed in Section 3.7.1 is implemented to calculate the cost of the paths, which can be embedded within the genetic algorithm to calculate the fitness function.

The fitness function $f(x)$ formulation problem can be described as follows: For a given $n \times m$ NoC architecture, calculating the sum of the costs of all possible paths that packets may travel through is required, where the cost represents the energy consumption of that path.

Algorithm 5 Calculate Fitness Algorithm

```
1: procedure CalculateFitness(tilesMatrix)
2:     tempX ← 0
3:     tempY ← 0
4:     fitnessCost ← 0
5:     for x = 0 to x ← numberOfRows − 1 do
6:         for y = 0 to y ← numberOfColumns − 1 do
7:             for tempX = 0 to x ← numberOfRows − 1 do
8:                 fittnessCost = fitnessCost + Dijkestra(tilesMatrix, tempX, x, y)
9:             end for
10:        end for
11:    end for
12:    fitnessCost ← 0
13:    for y = 0 to x ← numberOfColumns − 1 do
14:        for x = 0 to y ← numberOfRows − 1 do
15:            for tempY = 0 to y ← numberOfColumns − 1 do
16:                fittnessCost = fitnessCost + Dijkestra(tilesMatrix, tempY, x, y)
17:            end for
18:        end for
19:    end for
20:    return fitnessCost
21: end procedure
```

Referring to Algorithm 5, a code that loops through all the rows and columns in a two-dimensional mesh topology is defined. For example, the code starts with node $(x, y) = (0, 0)$ to calculate the energy cost of traversing packets from node (0,0) to every other nodes in the system. According to Dijkstra algorithm, if a path from node A to node B consumes less energy, it is said to be the shortest path.

In the algorithm presented, finding the shortest path from (0,0) to the next node (0,1) is done, and the cost of traversing packets from (0,0) to (0,1) is calculated. Then, the shortest path from node (0,0) to (0, 2) is determined, and cost of this path is accumulated with the previous path cost in an accumulator variable (e.g. $fitnessCost$ defined in Algorithm 5). The code keeps searching for paths from (0,0) until it reaches $(0, (n-1))$, where n is the number of rows/columns in a mesh topology. Next, the code runs to find costs of paths from (0,1) up to $(0, (n-1))$, and adds up the costs to the accumulator until all possible combinations of paths from source and destination are found. The total cost of all available paths in the network is acquired, which resembles the total energy consumed by the network and defining the $fitness$ of chromosome. A chromosome, which consumes less energy, is considered the fittest, therefore, it is a candidate to be selected as the sub-optimal solution for the problem.

3.7.6 Parents Selection

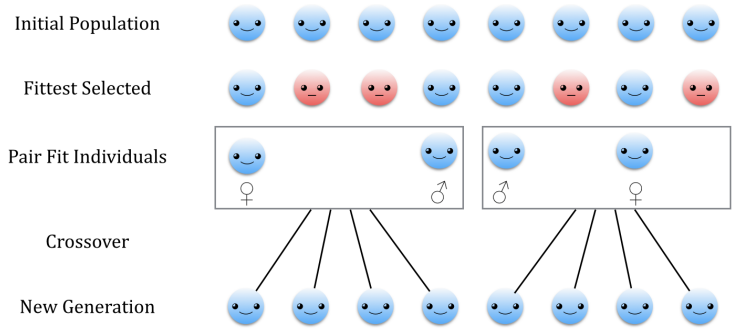


Figure 3.14: Parents Selection and Crossover

According to [23], the "Parents Selection" is the initial step for the crossover procedure. First, a set of the fittest individuals is selected from the initial population. The size of the *selectionset* determines how many children the crossover step need to produce in order to reach back to the size of the initial population (Refer to Equation 3.11). The example presented in Figure 3.14 generates a population of size 8 individuals, and sets the selection size to 4 discarding the 4 worst individuals in terms of fitness, therefore, each pair is required to crossover and produce $NumberofChildren = 8/(4/2) = 4$. In this approach, each pair of parents is equally likely to produce the same number of children to create fairness among the individuals.

$$PopulationSize = (SelectionSize/2) * NumberofChildren \tag{3.11}$$

3.7.7 Crossover

There are many types of crossover techniques that typically require a probability rate set, and allowing the chromosome pairs to be crossed over at random set points. The technique followed in this work uses the uniform crossover where bits are randomly selected from parents to produce the new children.

Figure 3.15 shows an example of 4×4 2D mesh architectures, where two parents are selected for crossover and the total allowed long-range links size is 14. Bits are selected simultaneously from parents assuming that equal link hops sizes from each chromosome are not always likely to be crossover. Some links in both parents may conflict with each other, because only one long-range-link is acceptable for every node. When copying different combinations of links to the children, further links cannot be added from the parents at some point and in some cases in order to reach the maximum allowed long-range-link hop size.

Algorithm 6 Crossover Algorithm

```
1: procedure CROSSOVER(TotalLinkSize, MaxCount)
2:     t ← 0                                              ▷ Allowed link hops to add
3:     counter ← 0                                        ▷ Checks for failure to select links
4:     while t ≠ totalLinksSize
       counter ≠ MaxCount do                              ▷ Max. allowed LRLs not reached
5:         link ← SelectLink(parentA)                     ▷ Select a long-range link from Parent(A)
6:         if
7:             link ≠ null then                           ▷ A Link with no Conflicts with is found
8:             if
9:                 child.Set(link) = true then            ▷ Connect Link to Child
10:                t ← t + link.size
11:            end if
12:        end if
13:        link ← SelectLink(parentB)                     ▷ Select a long-range link from Parent(B)
14:        if
15:            link ≠ null then                           ▷ A Link with no Conflicts with is found
16:            if
17:                child.Set(link) = true then            ▷ Connect Link to Child
18:                t ← t + link.size
19:            end if
20:        end if
21:        counter ← counter + 1
22:    end while
23:    if count = MaxCount then          ▷ Failed to add all required links using crossover
24:        while child.totalLinkSize ≠ TotalLinkSize do
25:            Mutate(child)             ▷ Mutate until Max. long-range-links are added
26:        end while
27:    end if
28: end procedure
```

In Figure 3.15, the algorithm managed to have *child*(2) hold exactly 7 link hops from each parent, while *child*(1) got only 5 link hops from *parent*(1) and 7 from *parent*(2). It is impossible to get another link from *parent*(1) to reach the desired result, so mutation was required to complete the procedure. (See Listing 6 for crossover details).

3.7.8 Mutation

The mutation is an operator that is used to create a kind of diversity between the different generations. In the mutation step, one or more genes is mutated or altered, so a "1" changes to "0" and a "0" changes to "1" causing the solution to change completely from the previous one in sometimes. The mutation normally has a mutation probability, which should be set as low as possible not to get to a random state of mutation. Usually, a value is generated for each gene to tell whether this bit shall be altered or not. This defines the most common type of mutation, which is called the single-point mutation. There are different types of mutation such as: bit string mutation, flip bit mutation, boundary mutation, uniform mutation, non-uniform mutation, and Gaussian mutation [23].

The mutation technique implemented in this work is the bit string mutation that allows a bit flips at random positions. Mutation is not always required for every individual, and is only requested when the total number of hop links did not reach the maximum in the crossover stage. Mutation flips some bits in order to allow the total number of hop links to reach the maximum.

3.7.9 Repeat

Once a child is produced in from crossover and mutation steps, the same steps the parent selection, crossover and mutation steps are repeated until the whole new population is gen-

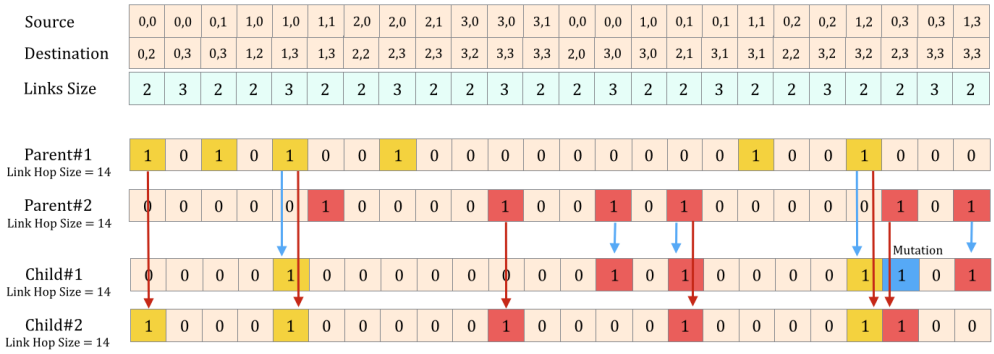


Figure 3.15: Crossover and Mutation Example

erated. A set of the fittest individuals from the previous generation is selected in order to produce new children for a generation population. Ultimately, the newer generation will show better genes than the ones in the older generation. Those better genes denote better long-range links selections. In this work, each two parents may produce one or more children, so depending on the number of fittest individuals selected and the population size, the number of children to produce is decided.

3.7.10 Termination

It is necessary to terminate the genetic algorithm when a good enough solution is found. In this work. The total number of generations and the number of individuals are set in the beginning of the program. Assuming that the number of generations is set to 20 and the number of individuals in each generation is a 100, the program will run to find the fittest architecture in the first generation. The information of the fittest architecture will be stored. Next, the program will run to find the fittest individual in the next generation and compare it with the previous fittest individual. If the new fittest individual from the second generation is fitter than the fittest individual from the first generation, the new individual will be saved as the fittest, otherwise, the first fittest information will remain the same. The program will continue running in the same manner until it reaches the maximum number of generations

set or if stopped manually.

Algorithm 7 Mutation Algorithm

```
1: procedure MUTATION(t, child)
2:     link ← SelectLink.Random()
3:     if link ∉ child.links then
4:         if child.getLink(link).getSource().getLRL() = null then
5:             if child.getLink(link).getDestination().getLRL() = null then
6:                 child.Connect(link)
7:                 allowedLinks = −link.Size
8:             end if
9:         end if
10:    end ifreturn allowedLinks
11: end procedure
```

Chapter 4

Simulation

After acquiring results from the genetic algorithm, the sub-optimal solution for the general purpose NoC with extra links is found. The next step is to test the NoC architecture via simulation to find how much energy the architecture will possibly consume. The NoC architecture with the extra links is directly implemented on Verilog and SystemVerilog to determine the energy efficiency of the new NoC design compared to the regular NoC.

4.1 Links and Routers Power Estimates

Since only one extra link is allowed per router, it means that the largest number of ports per router will be about 5 ports. This number of ports is particularity used in the central routers. Usually, 2D NoC synthesis papers have power estimates for routers of size 2-ports, 3-ports, and 4-ports, but not for the 5-ports. Therefore, power estimates from 3D NoCs are used, because vertical links or TSVs in a 3D NoC require adding an extra port to the router making it possible to have five ports in the central router. The average power estimates of three-dimensional NoC obtained from [37] are used. [37] synthesized the network implemented

for 70nm technology, and in this work the costs are substituted the Verilog code in order to signify the actual power consumed in a NoC by the routers and links. The average power estimates used from [37] are shown in Table 4.1 and Table 4.2. Accordingly, the costs of links and routers are calculated as displayed in Table 4.3.

Interconnect structure	Parameter			
	Electrical		Physical	
Horizontal bus	$\rho = 2.53$ μΩ × cm	$k_{ILD} = 2.7$	$w = 500$ nm	$s = 500$ nm
	$r_h = 46$ Ω/mm	$c_h = 192.5$ fF/mm	$t = 1{,}100$ nm	$h = 800$ nm
Vertical bus	$\rho = 5.65$ μΩ × cm	$r_v = 51.2$ Ω/mm	$w = 1{,}050$ nm	$L_{via} = 50$ μm
	$c_v = 600$ fF/mm			

Table 4.1: Links Power Estimates [37]

Ports (in × out)	2×2	3×2	3×3	4×3	4×4	5×4	5×5
Leakage power (W)	0.0069	0.0099	0.0133	0.0172	0.0216	0.0260	0.0319
Switching bit energy (pJ/bit)	0.3225	0.0676	0.5663	0.1080	0.8651	0.9180	1.2189

Table 4.2: Routers Power Estimates [37]

According to the mentioned power estimates in Table 4.1 and Table 4.2, the defined costs of the routers and links are set for simulating the NoC (Refer to Table 4.3).

Components	5×5 **mesh**	6×6 **mesh**	8×8 **mesh**
216 mm^2 mesh single-hop link cost	608.0	487.0	346.0
160 mm^2 mesh single-hop link cost	706.0	564.0	404.0
Repeaters (buffers)	150.0	150.0	150.0
2-Ports router	322.0	322.0	322.0
3-Ports router	566.0	566.0	566.0
4-Ports router	865.0	865.0	865.0
5-Ports router	1218.0	1218.0	1218.0

Table 4.3: Links, repeaters, and routers costs used in for Simulation

4.2 Basic Assumptions

Referring to Table 4.4, the simulation duration is set for all the different runs to be 10^4 cycles approximately, while it takes 2×10^3 cycles to warm-up. Since wormhole routing is used, it is assumed to have a flit of size 16 and a buffer of size 4 flits. In packet-switched networks, having relatively large buffers is avoided, because this can cause buffer bloating. Buffer bloating increases the latency and reduces the throughput of the network. Because of this issue, smaller buffers and larger flits are used. The sizes defined in this work are widely used in other researches, because they offer better performance and greater efficiency. The simulations used the uniform traffic model with a packet injection rate of 0.01 flits per cycle. VCs are used to avoid deadlocks in the network, and since VCs consume energy, the minimum possible number of VCs for each dimension in the two-dimensional mesh topology is used (Table 4.4).

Feature	**Description**
Simulation Duration	100000 cycles
Warm-up time	20000 cycles
Phit size	16 flits
Buffer size	4 flits
Virtual Channels	2 VCs
Injection Rate	0.01 flits/cycle
Traffic Model	Uniform traffic model

Table 4.4: Simulation Assumptions

4.3 NoC Simulation

Simulations for different sizes of mesh topologies are completed, such sizes are: 5×5, 6×6, and 8×8. Two different die sizes for each topology are used for the simulations. Normally, a die refers to the size of the CPUs or GPUs. In semiconductor technology, it is greatly favorable to reduce the die size as it carries several advantages. Smaller dies reduce the current to turn

transistors ON/OFF, which basically reduce the energy consumed and eventually reduce heat produced and enhance performance. Further, smaller die mean shorter channels consenting faster switching for the transistors.

As mentioned earlier, two different die sizes are used for simulating each topology: one bigger and one smaller die. The bigger die size's assumptions are based on Intel's i7 Core Sandy Bridge Micro-architecture processor, while the smaller die assumptions are based on Intel's i7 Core Ivy Bridge micro-architecture processor. Sandy Bridge-HE-4 of size 216 mm^2 holds about 1.16 billion transistors and uses 4 cores, while the Ivy Bridge-HE-4 has size of 160 mm^2 and carries 1.4 billion transistors connected through 4 cores. We had twelve simulations: two Sandy Bridge simulations and two Ivy Bridge simulations for each of the three topologies (5×5, 6×6, and 8×8).

4.3.1 5×5 NoC Simulation

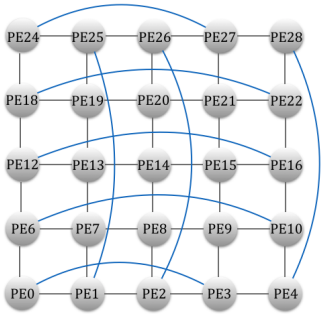


Figure 4.1: 5×5 Mesh with Extra Long-Range Links

The sub-optimal solution for the 5×5 mesh with extra links architecture acquired from the code is represented in Figure 4.1. In this architecture, adding high number of resources presented costing about 68% of the maximum possible resources. Results in Table 4.5 are based on the $216mm^2$ simulations, where in the first data column, results obtained from simulating the regular mesh are presented, while the second data column shows what has been obtained after adding the extra links to the network.

Topology	**Regular Mesh**	**Long Links Mesh**
Die Size(mm^2)	216	216
Link Size(mm)	≈ 3.67	≈ 3.67
Total Energy Consumption(nW)	1.58032×10^6	1.51407×10^6
Throughput($Mbps$)	1484	1478
Improvement (%)	-	-0.40%
Power/Throughput(bit/data)	1064.9	1024.4
Improvement	-	0.96

Table 4.5: Simulation for 216 mm^2 5×5 Mesh Topology

In terms of energy efficiency, about 4.4% of improvement is gained along with an inconsiderable 0.4% decrease in performance. On that grounds behalf, a factor that can provide a general performance measure for both throughput and energy consumption is defined. Equation 4.1 defines the factor implemented, and an improvement of 4.0% in energy/bit factor was gained comparatively to the Regular 216 mm^2 mesh.

Topology	**Regular Mesh**	**Long Links Mesh**
Die Size(mm^2)	160	160
Link Size(mm)	≈ 3.16	≈ 3.16
Total Energy Consumption(nW)	1.46423×10^6	1.40592×10^6
Throughput($Mbps$)	1484	1478
Improvement (%)	-	-0.4%
Power/Throughput(bit/data)	986.7	951.2
Improvement	0.93	0.89

Table 4.6: Simulation for 160 mm^2 5×5 Mesh Topology

$$\frac{Power}{Throughput} = \frac{Energy(nJ)/time(s)}{Data(Mbytes)/time(s)} = \frac{Energy(nJ)}{Data(Mbytes)} \tag{4.1}$$

Similarly, results displayed on Table 4.5 show simulation outcomes for the same mesh size(Figure 4.1) on a 160 mm^2 die, which is obviously, smaller that the 216 mm^2, have thinner wires, thus, will definitely show an improvement in terms of energy efficiency. The

160 mm^2 5×5 regular mesh topology showed an improvement of 7% over the same mesh simulated on 216 mm^2 die, while an improvement of 11% is gained when extra links were inserted to the network.

4.3.2 6×6 NoC Simulation

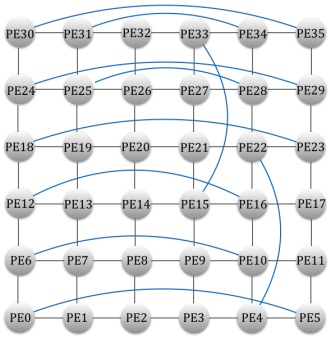


Figure 4.2: 6×6 Mesh with Extra Long-Range Links

Topology	**Regular Mesh**	**Long Links Mesh**
Die Size(mm^2)	216	216
Link Size(mm)	≈ 2.93	≈ 2.93
Total Energy Consumption(nW)	2.61384×10^6	2.23929×10^6
Throughput($Mbps$)	2149	2135
Throughput Improvement (%)	-	-0.65%
Power/Throughput	1216.3	1048.8
Improvement	-	0.89

Table 4.7: Simulation for 216 mm^2 6×6 Mesh Topology

For the 6×6 mesh network, 73% of total possible resources were added to the network, which is about 6% more than what had been used for the 5×5 topology. Results in Table 4.7 shows what has been obtained from simulations of the 216 mm^2 die. There are two simulations, one for the regular 6×6 mesh network, and the one is for the 6×6 mesh network with the extra links. Relatively to the regular mesh results, the outcomes presented an improvement of 11% in the energy/throughput factor for the mesh with extra links.

Topology	Regular Mesh	Long Links Mesh
Die Size(mm^2)	160	160
Link Size(mm)	≈ 2.53	≈ 2.53
Total Energy Consumption(nW)	2.44994×10^6	2.10437×10^6
Throughput($Mbps$)	2149	2135
Throughput Improvement (%)	-	-0.65%
Power/Throughput(bit/data)	1140.0	985.7
Improvement	0.94	0.81

Table 4.8: Simulation for 160 mm^2 6×6 Mesh Topology

On the other hand, the 160 mm^2 results showed a significant enhancement, where a gain of 19% was found for the mesh topology with extra links. In the 6×6 simulation, more resources are used than what has been used for the 5×5. 68% resources are used for the 5×5 network, while 73% are used for the 6×6. Some improvement is gained as more resources allows the presence of longer links, and longer links means longer paths consuming less energy than the formal paths in the regular mesh network. The regular mesh simulated for the 160 mm^2 die size showed an improvement of 6% over the 216 mm^2 die. The improvement is principally because of the smaller size and thinner wires as mentioned earlier.

4.3.3 8×8 NoC Simulation

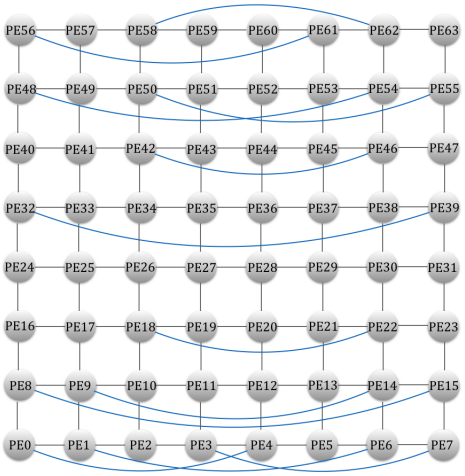


Figure 4.3: 8x8 Mesh Topology with Extra Long-Range Links

Topology	**Regular Mesh**	**Long Links Mesh**
Die Size(mm^2)	216	216
Link Size(mm)	≈ 2.1	≈ 2.1
Total Energy Consumption(nW)	5.97096×10^6	5.18414×10^6
Throughput($Mbps$)	3806	3804
Improvement (%)	-	-0.05%
Power/Throughput(bit/data)	1568.8	1362.8
Improvement	-	0.87

Table 4.9: Simulation for 216 mm^2 8×8 Mesh Topology

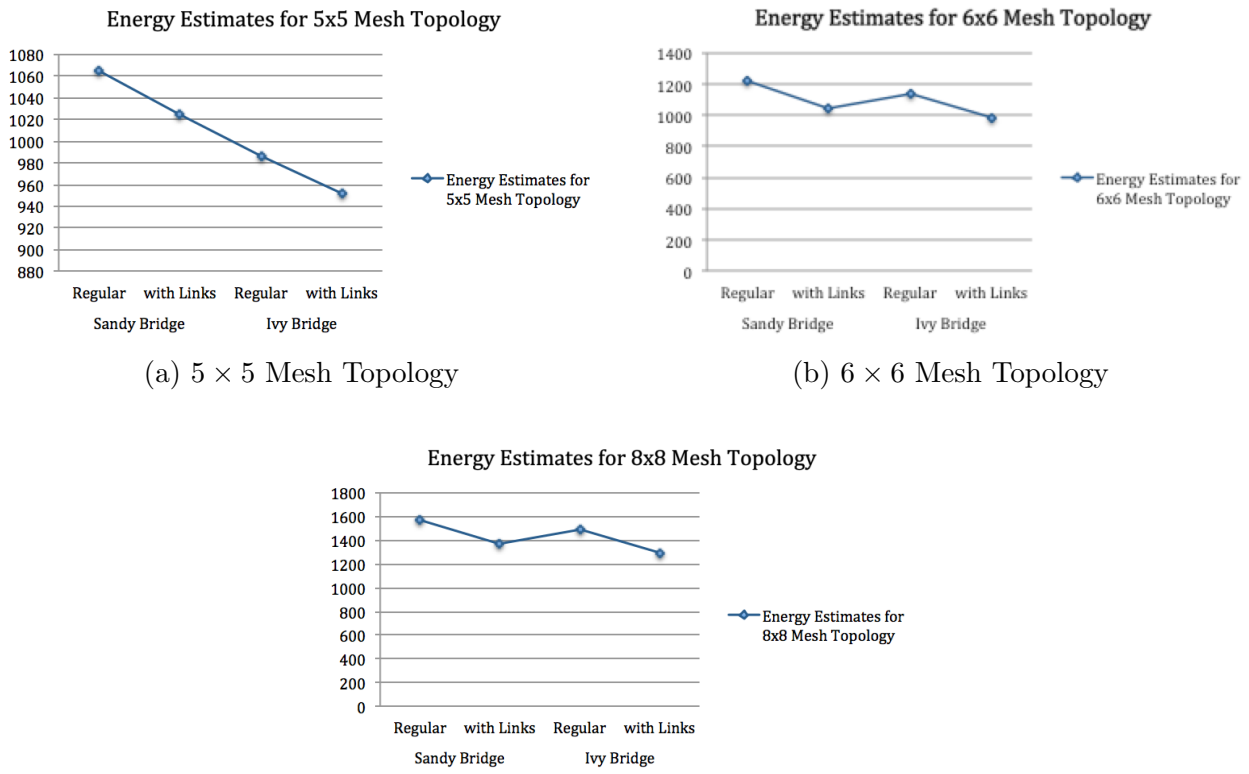
In the simulations of the 8×8 mesh network there was a valuable improvement when comparing the regular mesh architecture to the mesh with the extra links. In the 216 mm^2 die size, there was a gain of 13% in energy/throughput factor.

Topology	**Regular Mesh**	**Long Links Mesh**
Die Size(mm^2)	160	160
Link Size(mm)	≈ 1.8	≈ 1.8
Energy Consumption(nW)	5.66036×10^6	4.92396×10^6
Throughput($Mbps$)	3806	3804
Improvement (%)	-	-0.05%
Power/Throughput	1487.2	1294.4
Improvement	0.95	0.83

Table 4.10: Simulation for 160 mm^2 8×8 Mesh Topology

Comparatively, results obtained from the 160 mm^2 die simulations showed an improvement of 17% for the mesh architecture with the long-range links when using only 36% of total possible resources (Refer to Table 4.10). Throughput was relatively stable for the 8×8 mesh, where a negligible reduction of 0.04 was sighted. Comparing to other sizes of networks, it seems that the larger the network is, the greater energy efficiency we may gain when connecting extra links to the network. Further, increasing the number and extending the length of extra links we connect to the network improve the energy efficiency.

4.3.4 Results Analysis



(a) 5×5 Mesh Topology

(b) 6×6 Mesh Topology

(c) 8×8 Mesh Topology

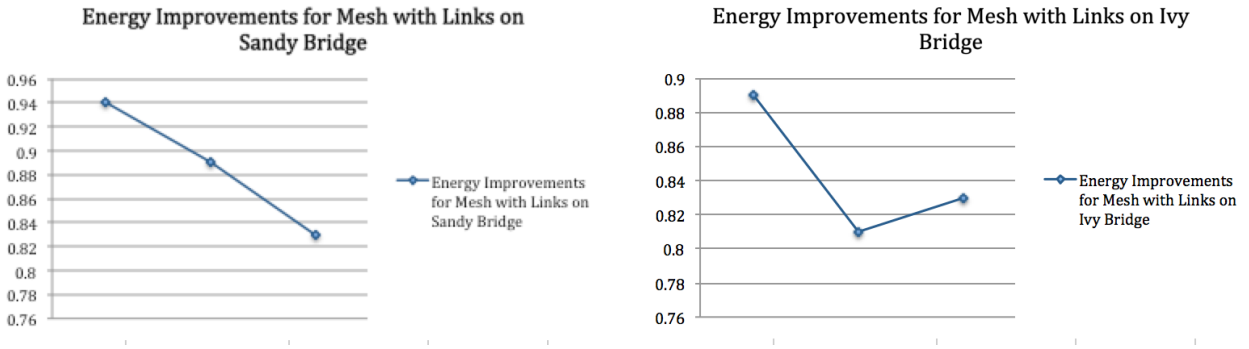
Figure 4.4: Energy Efficiency Improvements for 5x5, 6x6, and 8x8 Mesh Topology

Each sub-figure in Figure 4.4 show the improvements of energy efficiency using the power/throughput factor for every size of architecture. In Figure 4.4a, it is noticeable that the smaller the die and the more links connected, evident improvements in energy consumption are present. While in both of Sub-figure 4.4b, and 4.4c results shown similarity, as both are of smaller die size and have more links connected.

According to the basic assumptions of this work, the longer and the more links added, the better improvements are obtained. This is basically because longer links allow less hops though the high-energy consuming routers, while more hops through the low energy-consuming repeater. Moreover, allowing the use of more resources, means having a better chance to construct even longer links that helps satisfy the later. On the other hand, more

resources also mean more connections to the network, and more connections will ultimately lead to less energy-consuming paths. Results obtained showed that those assumptions are true, and also proved that the greater the size of the network, the better improvements is gained.

In Figure 4.5a results obtained for the 216 mm^2 are compared to each other for the three different sizes of simulated mesh topologies. Improvements have shown linear reduction in energy consumption as the network size goes larger. Conversely, in Figure4.5b, there was a drop in this linearity, and the energy consumption have increased a bit for the 8×8 comparing to the 6×6. Such increase is obviously because of using less number of resources for the 8×8 network.



(a) 160 mm^2 (b) 160 mm^2

Figure 4.5: 216 mm^2 and 160 mm^2 Simulation Results

Chapter 5

Conclusion

The steadfast developments in the silicon technologies reached to a point where we can sense billion-transistor chips; such single chip ICs archetypally include multiple complex heterogeneous constituents, such as, the programmable processors, on-chip memories, I/O interfaces, and communication architectures that functions mainly as an interconnection structure to serve different components' communications. The success of SoCs depends greatly on selecting the suitable design, using the right process technologies, and also, its ability to interconnect exiting modules in a plug-and-play fashion.

L. Benini and G. Michelli [4], presented a new approach to interconnect large number of intellectual properties using networking theories, which will help reduce power-related issues and get the upcoming chips to proceed along its march towards higher speed and greater efficiency. It is obvious how a NoC interconnection fabric carry great basis to improve the current SoC technologies. Presenting such complicated work requires huge efforts to develop a complete system that is aware of every possible breach within its components. It is evident that such interconnection acknowledge enhanced energy efficiency and boosted performance solutions. Thinking of supplementary techniques to suitably convalesce this technology is

mandatory, as far future computer needs necessitate even more energy efficient systems.

In this work, a way to improve the energy efficiency via adding links connecting distant nodes together is presented. The work is done initially by implementing Genetic Algorithms to a set of randomly generated architectures to find the near-optimal solution for the different sizes of two-dimensional mesh network topologies. Afterwards, the extra connections added to the NoC are implemented directly on pure Verilog and SystemVerilog to obtain the energy consumption information. It is found that connecting extra links to the mesh network generally improve energy efficiency. Throughout the twelve simulations performed in this work, the throughput and energy are calculated using as a factor of one another. An improvement of about 19% was gained for the NoC when connecting extra links to the 6×6 mesh network. The results obtained proved that adding long links to a two-dimensional mesh topology improve the energy efficiency of the NoC. Further, the more links inserted, the better improvements is obtained.

Future work is though about by considering the number of resources as a factor to improve the energy efficiency of a two-dimensional NoC. Moreover, leaning towards using realistic traffic models is one of the foreseen considerations for future work. There are also potential ideas for considering packet latency to obtain greater improvements.

Bibliography

[1] The law that's not a law. *Spectrum, IEEE*, 52(4):38–57, April 2015.

[2] W. Arden. Future roadblocks and solutions in silicon technology as outlined by the {ITRS} roadmap. *Materials Science in Semiconductor Processing*, 5(45):313 – 319, 2002. EMRS, symposium O.

[3] D. Beasley, D. R. Bull, and R. R. Martin. An overview of genetic algorithms: Part 1, fundamentals, 1993.

[4] L. Benini and G. De Micheli. Networks on chips: a new soc paradigm. *Computer*, 35(1):70–78, Jan 2002.

[5] L. Benini and G. De Michelli. *Networks on chips : technology and tools*. The Morgan Kaufmann series in systems on silicon. Elsevier Morgan Kaufmann Publishers, Amsterdam, Boston, Paris, 2006.

[6] P. E. Berman, L. Gravano, G. D. Pifarré, and J. L. C. Sanz. Adaptive deadlock- and livelock-free routing with all minimal paths in torus networks. In *Proceedings of the Fourth Annual ACM Symposium on Parallel Algorithms and Architectures*, SPAA '92, pages 3–12, New York, NY, USA, 1992. ACM.

[7] W. Brinkman, D. Haggan, and W. Troutman. A history of the invention of the transistor and where it will lead us. *Solid-State Circuits, IEEE Journal of*, 32(12):1858–1865, Dec 1997.

[8] M. Cahay and S. Bandyopadhyay. An electron's spin—part i. *Potentials, IEEE*, 28(3):31–35, May 2009.

[9] L. P. Carloni, K. L. McMillan, and A. L. Sangiovanni-Vincentelli. Theory of latency-insensitive design. *Trans. Comp.-Aided Des. Integ. Cir. Sys.*, 20(9):1059–1076, Nov. 2006.

[10] V. Chandra, A. Xu, H. Schmit, and L. Pileggi. An interconnect channel design methodology for high performance integrated circuits. In *Proceedings of the Conference on Design, Automation and Test in Europe - Volume 2*, DATE '04, pages 21138–, Washington, DC, USA, 2004. IEEE Computer Society.

[11] E. Conrad, S. Misenar, and J. Feldman. *CISSP Study Guide*. Syngress Publishing, 2 edition, 2012.

[12] W. Dally and B. Towles. *Principles and Practices of Interconnection Networks*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2003.

[13] E. W. Dijkstra. A note on two problems in connexion with graphs. *Numer. Math.*, 1(1):269–271, Dec. 1959.

[14] B. A. Floyd, C.-M. Hung, and K. K. O. Intra-chip wireless interconnect for clock distribution implemented with integrated antennas, receivers, and transmitters. *IEEE Journal of Solid-State Circuits*, 37(5):543–552, May 2002.

[15] H. Fukś and A. T. Lawniczak. Performance of data networks with random links. *Math. Comput. Simul.*, 51(1-2):101–117, Dec. 1999.

[16] H. Fuks and A. T. Lawniczak. Performance of data networks with random links. *CoRR*, adap-org/9909006, 1999.

[17] A. Ganguly, K. Chang, S. Deb, P. P. Pande, B. Belzer, and C. Teuscher. Scalable hybrid wireless network-on-chip architectures for multicore systems. *IEEE Transactions on Computers*, 60(10):1485–1502, Oct 2011.

[18] C. J. Glass and L. M. Ni. The turn model for adaptive routing. *J. ACM*, 41(5):874–902, Sept. 1994.

[19] K. Goossens, O. P. Gangwal, J. Röver, and A. P. Niranjan. Interconnect and memory organization in SOCs for advanced set-top boxes and TV — evolution, analysis, and trends. In J. Nurmi, H. Tenhunen, J. Isoaho, and A. Jantsch, editors, *Interconnect-Centric Design for Advanced SoC and NoC*, chapter 15, pages 399–423. Kluwer, Apr. 2004.

[20] P. Guerrier and A. Greiner. A generic architecture for on-chip packet-switched interconnections. In *Proceedings of the Conference on Design, Automation and Test in Europe*, DATE '00, pages 250–256, New York, NY, USA, 2000. ACM.

[21] J. Kleinberg. The small-world phenomenon: An algorithmic perspective. In *Proceedings of the Thirty-second Annual ACM Symposium on Theory of Computing*, STOC '00, pages 163–170, New York, NY, USA, 2000. ACM.

[22] A. Mello, L. Tedesco, N. Calazans, and F. Moraes. Virtual channels in networks on chip: Implementation and evaluation on hermes noc. In *Proceedings of the 18th Annual Symposium on Integrated Circuits and System Design*, SBCCI '05, pages 178–183, New York, NY, USA, 2005. ACM.

[23] M. Mitchell. *An Introduction to Genetic Algorithms*. MIT Press, Cambridge, MA, USA, 1998.

[24] G. E. Moore. Cramming more components onto integrated circuits, reprinted from electronics, volume 38, number 8, april 19, 1965, pp.114 ff. *Solid-State Circuits Society Newsletter, IEEE*, 11(5):33–35, Sept 2006.

[25] G. E. Moore, D. C, and Brock. *Understanding Moore's Law, four decades of innovation*. Chemical Heritage Foundation, 2006.

[26] M. Newman and D. Watts. Scaling and percolation in the small-world network model. *Physical Review E*, 60(6):7332, 1999.

[27] U. Y. Ogras and R. Marculescu. "it's a small world after all": Noc performance optimization via long-range link insertion. *IEEE Trans. Very Large Scale Integr. Syst.*, 14(7):693–706, July 2006.

[28] G. O'Regan. *A Brief History of Computing*. Springer Publishing Company, Incorporated, 2nd edition, 2012.

[29] S. Pasricha and N. Dutt. *On-Chip Communication Architectures: System on Chip Interconnect*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2008.

[30] B. Randell. *The Origins of Digital Computers: Selected Papers (Monographs in Computer Science)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 1982.

[31] R. Saleh, S. Wilton, S. Mirabbasi, A. Hu, M. Greenstreet, G. Lemieux, P. Pande, C. Grecu, and A. Ivanov. System-on-chip: Reuse and integration. *Proceedings of the IEEE*, 94(6):1050–1069, June 2006.

[32] M. B. Stensgaard and J. Spars. Renoc: A network-on-chip architecture with reconfigurable topology. In *Networks-on-Chip, 2008. NoCS 2008. Second ACM/IEEE International Symposium on*, pages 55–64, April 2008.

[33] K. Tatas, K. Siozios, D. Soudris, and A. Jantsch. *Designing 2D and 3D Network-on-chip Architectures*. Springer, 2014.

[34] D. J. Watts. *Small worlds: The dynamics of networks between order and randomness*. Princeton University Press, Princeton, NJ, 1999.

[35] D. J. Watts and S. H. Strogatz. Collective dynamics of /'small-world/' networks. *Nature*, 393(6684):440–442, june 1998.

[36] L. Xiu. *The Big Picture*, pages 1–16. Wiley-IEEE Press, 2008.

[37] S. Yan and B. Lin. Design of application-specific 3d networks-on-chip architectures. In *Computer Design, 2008. ICCD 2008. IEEE International Conference on*, pages 142–149, Oct 2008.